



Universidad de San Carlo de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Introducción a la programación y computación 1
Ing. Moisés Velásquez
Auxiliar. Pablo Oliva

PROYECTO No.2: MANUAL TECNICO

Guillermo Enrique Marroquin Morán

202103527

Guatemala 24 de octubre. de 25

INDICE

| | |
|--|----------|
| MANUAL TÉCNICO | 4 |
| REQUERIMIENTOS MÍNIMOS DEL PROGRAMA | 4 |
| EXPLICACIÓN DE LAS PARTES DEL PROYECTO | 5 |
| LIBRERÍAS USADAS | 5 |
| EXPLICACION DEL CODIGO | 7 |
| CONTRUCTORES: | 7 |
| <i>Contructor de usuarios</i> | 7 |
| <i>Contructor de vendedores</i> | 8 |
| <i>Contructor de clientes</i> | 9 |
| <i>Contructor de productos</i> | 9 |
| <i>Contructor de productos alimenticios (ProductoComida)</i> | 10 |
| <i>Contructor de productos tecnologicos (ProductoTec)</i> | 10 |
| <i>Contructor de productos otros (ProductoOtros)</i> | 11 |
| <i>Contructor de pedidos</i> | 11 |
| <i>Contructor de Carrito de compras</i> | 12 |
| ADMINISTRADOR DE USUARIOS | 12 |
| <i>Carga de Usuarios</i> | 13 |
| <i>Guardar Usuarios</i> | 13 |
| <i>Autenticacion de Usuarios</i> | 13 |
| <i>Agregar Usuarios</i> | 14 |
| <i>Codigo Repetido</i> | 14 |
| <i>Actualizar Usuarios</i> | 14 |
| <i>Eliminar usuarios</i> | 15 |
| ADMINISTRADOR DE VENDEDORES | 15 |
| <i>Cargar Vendedores</i> | 16 |
| <i>Cargar Vendedores</i> | 16 |
| <i>Codigo Repetido</i> | 17 |
| <i>Creacion de Vendedor</i> | 17 |
| <i>Modificacion de Vendedor</i> | 17 |
| <i>Eliminacion de Vendedor</i> | 18 |
| <i>Validacion de Vendedor</i> | 18 |
| <i>Datos para la tabla de Vendedor</i> | 18 |
| <i>Get Lista de Vendedores</i> | 19 |
| ADMINISTRADOR DE PRODUCTOS | 19 |
| <i>Cargar Productos</i> | 19 |
| <i>Guardar Productos</i> | 20 |
| <i>Buscar Productos y codigo repetido</i> | 20 |
| <i>Creación de productos</i> | 21 |
| <i>Modificación de productos</i> | 21 |
| <i>Eliminación de productos</i> | 22 |
| <i>Carga de productos</i> | 22 |
| <i>Datos de tabla de productos</i> | 23 |
| <i>Agregar Stock</i> | 23 |
| <i>Registrar Stock</i> | 24 |
| <i>Datos de la tabla de cliente y y reservar de Stock.</i> | 24 |
| ADMINISTRADOR DE PEDIDOS | 25 |
| <i>Datos de la tabla</i> | 25 |

| | |
|---|-----------|
| <i>Crear pedido</i> | 26 |
| <i>Confirmación de pedido</i> | 26 |
| ADMINISTRADOR DE COMPRAS | 27 |
| <i>Carga de compras</i> | 27 |
| <i>Guardar Compra</i> | 28 |
| <i>Agregar compra</i> | 28 |
| <i>Datos de la tabla del historial de compras</i> | 29 |
| <i>Buscar actualizaciones de Stock</i> | 30 |
| ADMINISTRADOR DE CLIENTES | 30 |
| <i>Cargar clientes</i> | 31 |
| <i>Guardar clientes Y código repetido</i> | 32 |
| <i>Creación de clientes</i> | 33 |
| <i>Tabla de datos de cliente y buscar cliente</i> | 33 |
| <i>Modificar cliente</i> | 34 |
| <i>Eliminar cliente</i> | 34 |
| <i>Cargar clientes</i> | 35 |

Manual Técnico

Requerimientos mínimos del programa.

Para instalar apache NetBeans los requisitos mínimos son:

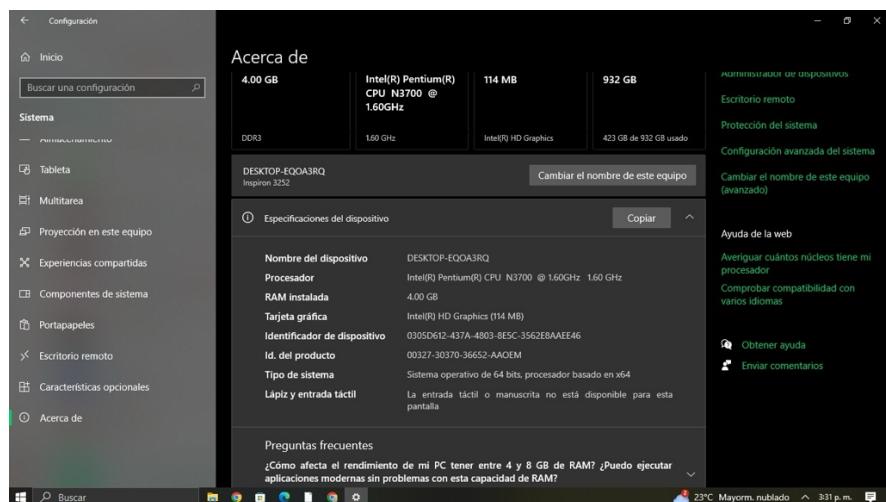
- 781 MB de Espacio Libre en el Disco Duro.
- 512 MB de RAM.
- Procesador Intel Pentium III a 800 MHz.
- Compatible con Windows, macOS y Linux.

(Janl, 2022)

Requisitos del ordenador usado para el proyecto:



Requisitos del ordenador con sistema Windows:



Explicación de las partes del proyecto.

Librerías usadas

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintWriter;
```

Liberias para los archivos:

- File: Para representar las rutas de los archivos en nuestro caso son los archivos .csv, .ser, y .pdf.
- FileReader, BufferedReader: Para leer archivos de texto, línea por línea se usaron para cargar datos desde el csv.
- FileWriter, PrintWriter: Para escribir en los archivos de texto, se usaron para guardar datos en CSV y la bitácora.
- FileOutputStream, ObjectOutputStream: Para guardar el estado de los objetos como las listas de productos, vendedores, etc. en archivos .ser.
- FileInputStream, ObjectInputStream: Para cargar el estado de los objetos desde archivos .ser.
- Serializable: Interfaz marcador utilizada en las clases de datos Usuarios, Productos, Pedidos, etc, nos permite que sus objetos sean cargados.
- IOException, FileNotFoundException: Para manejar errores relacionados con la lectura o escritura de archivos.

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPasswordField;//xd
import javax.swing.JTextField;
import javax.swing.SwingConstants;
import javax.swing.BorderFactory;
import javax.swing.JFileChooser;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
```

Librerías de interfaz gráfica:

- JFrame: La ventana principal para cada parte visual como los menús o los formularios.
- JPanel: Contiene todo lo que se vaya a organizar a otros componentes dentro de una ventana.
- JLabel: Para mostrar texto en las ventanas.
- JTextField, JPasswordField: Campos para que el usuario ingrese texto como el código, los nombres, y la contraseña.
- JButton: Botones en los que el usuario hace clic para realizar acciones.
- JComboBox: Es una lista desplegable.
- JTable, DefaultTableModel: Para mostrar datos en formato de tabla como la lista de vendedores, de productos, de clientes.
- JScrollPane: Para añadir barras para desplazarse a componentes grandes como tablas o áreas de texto.
- JTextArea: Área para mostrar texto de múltiples líneas se usó en ventanas de carga masiva para mostrar resultados.
- JOptionPane: Para mostrar mensajes emergentes simples información, mensajes de error, confirmaciones opciones.
- JFileChooser: Para permitir al usuario seleccionar archivos del sistema se usó en cargas masivas).
- SwingConstants: Constantes para definir alineación del texto.
- BorderFactory: Para añadir bordes a los paneles.
- BorderLayout, GridLayout, GridBagLayout, GridBagConstraints, FlowLayout: Administradores de diseño definen cómo se posicionan y se redimensiona los componentes dentro de un panel o una ventana.
- Font: Para definir el tipo de letra, estilo y tamaño del texto en los componentes.
- Insets: Usado con GridBagLayout para definir márgenes alrededor de los componentes.

```
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter; import java.time.temporal.ChronoUnit;/
    import java.time.format.DateTimeParseException;
```

Librerías del tiempo

- LocalDateTime: Es una representación para una fecha y una hora específicas
- LocalDate: Representa solo una fecha.
- DateTimeFormatter: Para convertir objetos de fecha y hora a texto con un formato específico como el que nos pidieron: "dd/MM/yyyy HH:mm:ss" y viceversa convertir de texto a fecha y hora.
- format.DateTimeParseException: Excepción específica que ocurre si una cadena de texto no coincide con el formato de fecha esperado.

- ChronoUnit: Permite calcular fácilmente la diferencia entre dos fechas en unidades específicas, se uso para calcular los dias restantes para un producto por caducar.

```
import com.itextpdf.kernel.pdf.PdfDocument;
import com.itextpdf.kernel.pdf.PdfWriter;
import com.itextpdf.layout.Document;
import com.itextpdf.layout.element.Cell;
import com.itextpdf.layout.element.Paragraph;
import com.itextpdf.layout.element.Table;
import com.itextpdf.layout.property.UnitValue;
import java.io.FileNotFoundException;
import com.itextpdf.layout.property.TextAlignment; /
```

ItextCore

- PdfWriter: Especifica el archivo de salida donde se escribirá el dpcumento pdf.
- PdfDocument: Representa la estructura interna del documento pdf.
- Document: Nos facilita la adición de contenido al documetno pdf.
- Paragraph: Para añadir bloques de texto al documento
- Table: Para crear tablas estructuradas dentro del pdf.
- Cell: Representa una celda dentro de una tabla.
- UnitValue: Usado para definir anchos de las columnas de la tabla.
- TextAlign: Enumeración para especificar la alineación del texto.

EXPLICACION DEL CODIGO.

CONTRUCTORES:

Contructor de usuarios

```
public class Usuarios implements Serializable {
    // id de serie, me lo dijo yutu :((

    private static final long serialVersionUID = 1L;

    //los campos de todos los usuarios
    public static final int ID = 0;
    public static final int NOMBRE = 1;
    public static final int CONTRASEÑA = 2;
    public static final int TIPOUSUR = 3;
    public static final int CAMPOS = 4;

    protected String id;
    protected String nombre;
    protected String Contraseña;
    protected String tipoUsuario;

    public Usuarios(String id, String nombre, String Contraseña, String tipoUsuario) {
        //gracias inge moises :3
        this.id = id;
        this.nombre = nombre;
        this.Contraseña = Contraseña;
        this.tipoUsuario = tipoUsuario;
    }

    if (this.tipoUsuario != null && !this.tipoUsuario.trim().isEmpty()) {
        this.tipoUsuario = this.tipoUsuario.trim().toUpperCase();
    } else {
        this.tipoUsuario = "ADMIN"; //asginamos un admin por si las moscas
    }

    //los getters, GRACIAS INGE ME ESTA SALVANDO LA COLA
    public String getId() {
        return id;
    }

    public String getNombre() {
        return nombre;
    }

    public String getcontraseña() {
        return Contraseña;
    }

    public String getTipoUsuario() {
        return tipoUsuario;
    }
}
```

Éste constructor fue creado para contener y crear a los usuarios, cómo se puede ver se se ha creado diferentes constantes públicas y variables protegidas que representan los atributos del usuario como

MANUAL TECNICO

el nombre el código la contraseña el tipo de usuario y la última constante representa los campos para que el siguiente constructor los tome en cuenta, seguido se crea los getters y setter de ciertos atributos que no facturan serán de utilidad, por último hasta el método TR que nos permite obtener los datos como un arreglo.

Constructor de vendedores

```

public class Vendedor extends Usuarios {

    // CONSTANTES DE POSICIÓN DE CAMPOS EN CSV
    public static final int ID = Usuarios.ID;
    public static final int NOMBRE = Usuarios.NOMBRE;
    public static final int CONTRASEÑA = Usuarios.CONTRASEÑA;
    public static final int TIPOUSUARIO = Usuarios.TIPOUSUARIO; // "VENDEDOR"

    public static final int GENERO = Usuarios.CAMPOS; // El índice empieza después del último campo de Usuarios
    public static final int VENTAS_HECHAS = Usuarios.CAMPOS + 1;
    public static final int NUM_CAMPOS_VENDEDOR = Usuarios.CAMPOS + 2; // Total de campos para el CSV

    // Atributos de la claseeee
    private String genero;
    private int Ventashechas;
    private double TotalVentasHechas; // el nombre lo dice todo no?

    // El constructor
    public Vendedor(String id, String nombre, String Contraseña, String genero) {
        super(id, nombre, Contraseña, "VENDEDOR");

        this.genero = genero;
        this.Ventashechas = 0;
        this.TotalVentasHechas = 0.0;
    }

    // GETTERS Y SETTERS, y gracias inge moiseeee, me salvo la cola
    public String getId() {
        return super.id;
    }

    public String getGenero() {
        return genero;
    }

    @Override
    public String[] toArray() {
        // Devuelve el array completo eñ ID, NOMBRE, CONTRASEÑA, TIPO, GÉNERO, VENTAS
        return new String[]{
            super.id,
            super.nombre,
            super.Contraseña,
            super.tipoUsuario,
            this.genero,
            String.valueOf(this.Ventashechas)
        };
    }

    public String bienvenidaV() {
        return "bienvenid@ al modulo de vendedor de Hip Shop, " + super.nombre;
    }
}

public int getVentasHechas() {
    return Ventashechas;
}

public void setVentasHechas(int ventas) {
    this.Ventashechas = ventas;
}

public String setId() {
    return id;
}

public void setNombre(String nombre) {
    super.nombre = nombre;
}

public void setContraseña(String Contraseña) {
    super.Contraseña = Contraseña;
}

@Override
public String getNombre() {
    return super.nombre;
}

public String getContraseña() {
    return super.Contraseña;
}

public double getTotalVentasHechas() {
    return TotalVentasHechas;
}

public void setTotalVentasHechas(double TotalVentasHechas) {
    this.TotalVentasHechas = TotalVentasHechas;
}

```

Continuando con el constructor de vendedores, hereda características de la clase usuarios cómo se podrá ver en la imagen todos los atributos vienen derivados de la clase de usuarios, como modificación se le agregó 3 constantes más, que representan al género las ventas hechas el número de campos del vendedor, seguido se creó variables usadas en la clase, qué representan estos tres últimos atributos de igual manera se obtienen sus respectivos, setters y getters para obtener y dichos datos, cómo últimos datos se obtiene el metro String torre que nos devuelve completo el arreglo como último se agregó un mensaje de bienvenida.

Constructor de clientes

```
public class Cliente extends Usuarios {
    //atributos unicos del cliente
    private String genero;
    private String cumpleaños;

    public Cliente(String id, String nombre, String contraseña, String genero, String cumpleaños){
        super(id,nombre, contraseña, "CLIENTE");
        this.genero = genero;
        this.cumpleaños = cumpleaños;
    }
    //los getters
    public String getGenero(){
        return genero;
    }
    public String getCumpleaños(){
        return cumpleaños;
    }
    @Override
    public String getNombre(){
        return nombre;
    }

    public String getContraseña(){
        return Contraseña;
    }
    //los setters para modificar las cosas
    public void setGenero(String genero){//SOLO DOS GENEROS ALV
        this.genero = genero;
    }
    public void setCumpleaños(String cumpleaños){
        this.cumpleaños = cumpleaños;
    }
    public void setNombre(String nombre){
        this.nombre = nombre;
    }
}
```

De igual manera que el constructor de vendedores, el constructor de clientes hereda atributos de la clase de usuarios sólo que esta vez se le agrega un parámetro extra que es el cumpleaños que parámetro del género también lo incluye el vendedor de igual manera se agrega los, getters y setters para obtener los datos y cargarlos.

Constructor de productos

```
public abstract class Productos implements Serializable {
    //las constantes de posicion
    public static final int CODIGO =0;
    public static final int NOMBRE =1;
    public static final int PRECIO =2;
    public static final int STOCK =3;
    public static final int CATEGORIA =4;
    public static final int CAMPOS_Prod =6;
    //usamos los protected para usarlas en las demás clases
    protected String codigo;
    protected String nombre;
    protected String categoria;
    protected double precio;
    protected int stock;
}

public Productos(String codigo, String nombre, String categoria, double precio, int stock){
    this.codigo = codigo;
    this.nombre = nombre;
    this.categoria = categoria;
    this.precio = precio;
    this.stock = stock;
}

//los getters ya saben lo que dire a continuacion... gracias inge xd
public String getCodigo(){
    return codigo;
}
public String getNombre(){
    return nombre;
}
public String getCategoría(){
    return categoria;
}
public double getPrecio(){
    return precio;
}
public int getStock(){
    return stock;
}
public void setNombre(String nombre){
    this.nombre = nombre;
}

public void setStock(int stock){
    this.stock = stock;
}

//los metodos abstractos para los gestores, asi no la kgo como con los vendedores
public abstract String[] toArray();
public abstract String toCsLine();
```

Éste constructor se le crearon diferentes constantes que representan en el código, nombre, precio, Stock, categoría y como últimos se le agrega los campos del producto, seguido se le crean variables protegidas que representan el código, el nombre, la categoría, el precio, el Stock de igual manera si le agregan los setters y getters para obtener y cargar datos.

Constructor de productos alimenticios (ProductoComida)

```
public class productoComida extends Productos {
    //mas constantes
    public static final int VENCIMIENTO = Productos.CAMPOS_Prod;
    public static final int CAMPOCOM = Productos.CAMPOS_Prod + 1;

    private String FechaVencer;
    public productoComida(String codigo, String nombre, double precio, int stock, String FechaVencer) {
        super(codigo, nombre, "ALIMENTO", precio, stock);
        this.FechaVencer = FechaVencer;
    }
    public String getFechaVencer(){
        return FechaVencer;
    }

    @Override
    public String[] toArray() {
        return new String[]{
            super.codigo,
            super.nombre,
            super.categoría,
            String.valueOf(super.precio),
            String.valueOf(super.stock),
            this.FechaVencer
        };
    }
    @Override
    public String toCsLine() {
        return super.codigo + "," + super.nombre + "," + super.precio + "," + super.stock + "," + super.categoría + "," + this.FechaVencer;
    }
    public void setFechaVencer(String FechaVencer){
        this.FechaVencer = FechaVencer;
    }
}

public void setFechaVencer(String FechaVencer) {
    this.FechaVencer = FechaVencer;
}

public static boolean FechaValida (String fechaStr) {
    if(fechaStr == null || fechaStr.trim().isEmpty()){
        return false;
    }
    try{
        DateTimeFormatter formato = DateTimeFormatter.ofPattern ("yyyy-MM-dd");
        LocalDate fecha = LocalDate.parse (fechaStr.trim(), formato);
        return true;
    } catch(DateTimeParseException e){
        return false;
    }
}
```

Esta clase hija hereda atributos de su clase padre, la diferencia es de que este constructor agrega una tributo específico cuál representa la fecha de caducidad y el campo que representa al producto alimenticio, se agregan los setters y getters para cargar y obtener datos de última manera o sea si son pequeño método para verificar la fecha de caducidad.

Constructor de productos tecnologicos (ProductoTec)

```
public class productoTec extends Productos {

    public static final int GARANTIA = Productos.CAMPOS_Prod;
    public static final int CAMPOSTEC = Productos.CAMPOS_Prod + 1;

    private int MesesGarantia;

    public productoTec(String codigo, String nombre, double precio, int stock, int MesesGarantia) {
        super(codigo, nombre, "TECNOLOGIA", precio, stock);
        this.MesesGarantia = MesesGarantia;
    }

    public int getMesesGarantia() {
        return MesesGarantia;
    }

    public void setMesesGarantia(int MesesGarantia) {
        this.MesesGarantia = MesesGarantia;
    }

    //ets madre solo al aplicar los metodos abstractos

    @Override
    public String[] toArray() {
        return new String[]{
            super.codigo,
            super.nombre,
            super.categoría,
            String.valueOf(super.precio),
            String.valueOf(super.stock),
            String.valueOf(this.MesesGarantia)
        };
    }

    @Override
    public String toCsLine() {
        return super.codigo + "," + super.nombre + "," + super.precio + "," + super.stock + "," + super.categoría + "," + this.MesesGarantia;
    }
}
```

Esta clase hija también hereda muchos de los atributos de la clase padre, con la diferencia de qué éste tiene los meses de garantía y respectivo campo que representa a los productos tecnológicos, se agregan los set y get de los meses de garantía para obtenerlos y cargarlos cuando sea necesario.

Constructor de productos otros (ProductoOtros)

```
public class productoOtros extends Productos {  
    //total de campos para el csv de otros productos  
  
    public static final int CAMPORT = Productos.CAMPOS_Prod;  
  
    public productoOtros(String codigo, String nombre, double precio, int stock){  
        super(codigo, nombre, "OTROS", precio, stock);  
    }  
  
    @Override  
    public String[] toArray(){  
        return new String[]{  
            super.codigo,  
            super.nombre,  
            super.categoría,  
            String.valueOf(super.precio),  
            String.valueOf(super.stock),  
            "N/A" //usamos eso para mantener la consistencia en las tablas  
        };  
    }  
  
    @Override  
    public String toCsvLine(){  
        return super.codigo + "," + super.nombre + "," + super.precio + "," + super.stock + "," + super.categoría + "," + "N/A";  
    }  
}
```

Al igual que la casa padre esta clase hija eres a todos los atributos sólo con la diferencia de que aquí se le va a agregar el campo respectivo a los productos que no pertenezcan categoría alimento o tecnología.

Constructor de pedidos

```
public class Pedidos implements Serializable {  
  
    private String IDpedido;  
    private transient LocalDateTime fechaGeneracion;  
    private String IDcliente;  
    private String NombreCliente;  
    private double total;  
    private ProdCarrito[] Prods; //para guardar productos  
    private int cantProductos;  
    private String fechaGeneracionStr; // guardar fecha como string  
  
    //para los pedidos, el constructor  
    public Pedidos(String IDpedido, LocalDateTime fecha, String IDcliente, String NombreCliente, double total, ProdCarrito[] Prods, int cantProductos) {  
        this.IDpedido = IDpedido;  
        this.fechaGeneracion = fecha;  
        this.IDcliente = IDcliente;  
        this.NombreCliente = NombreCliente;  
        this.total = total;  
        this.Prods = Prods;  
        this.cantProductos = cantProductos;  
        this.fechaGeneracionStr = fecha.toString();  
    }  
    //los guiones o como se diga del inglés al español alv  
    public String getIDpedido(){  
        return IDpedido;  
    }  
  
    public LocalDateTime getFechaGeneracion(){  
        if(fechaGeneracion == null && fechaGeneracionStr != null){  
            try{  
                fechaGeneracion = LocalDateTime.parse(fechaGeneracionStr);  
            } catch(Exception e){  
                //nada xd  
            }  
        }  
        return fechaGeneracion;  
    }  
  
    public String getIdCliente(){  
        return IDcliente;  
    }  
  
    public String getNombreCliente(){  
        return NombreCliente;  
    }  
  
    public double getTotal(){  
        return total;  
    }  
  
    public int getCantProductos(){  
        return cantProductos;  
    }  
  
    public ProdCarrito[] getProds(){  
        return Prods;  
    }  
}
```

Éste constructor se le crearon diferencia atributos privados y corresponden al código del pedido, la fecha de generación del mismo, el código del cliente, el nombre del cliente, el total de compra, la cantidad de productos, y se agregó dos atributos adicionales para guardar los productos y la fecha de generación en cadena de texto, se agregaron los setters y getters para obtener y cargar datos como última distancia se creó un método para la fecha de generación del pedido para así convertirla de fecha a una cadena de texto.

Constructor de Carrito de compras

```
public class Carrito {  
    private static final int MaxProdCarro = 50; //un maximo de 50 productos distintos en el carro, porque no me lo voy a poder muy especialito con esto  
    private static ProdCarrito[] Prod = new ProdCarrito[MaxProdCarro];  
    private static int contProductos = 0;  
  
    public static ProdCarrito[] getProd() {  
        return Prod;  
    }  
  
    public static int getContProd() {  
        return contProductos;  
    }  
  
    //un metodo para agregar un producto al carro  
    public static boolean AgregarProducto(Products producto, int cantidad) {  
        if(contProductos >= MaxProdCarro) {  
            JOptionPane.showMessageDialog(null, "Has alcanzado el limite de productos en el carro", "Carro de compras lleno", JOptionPane.WARNING_MESSAGE);  
            return false;  
        }  
  
        Prod[contProductos] = new ProdCarrito(producto, cantidad);  
        JOptionPane.showMessageDialog(null, cantidad + " unidades de " + producto.getNombre() + " agregado al carro de compras", "Tarea Exitosa", JOptionPane.INFORMATION_MESSAGE);  
        return true;  
    }  
  
    //para el boton de eliminar  
    public static void EliminarProd(int indiceProd) {  
        if(indiceProd < 0 || indiceProd > contProductos) return; //por si acaso  
        for(int i = indiceProd; i < contProductos - 1; i++) {  
            Prod[i] = Prod[i+1];  
        }  
        Prod[contProductos-1] = null;  
        contProductos--;  
    }  
  
    public static boolean actualizarCont (int indiceProd, int Ncantidad) {  
        if(indiceProd < 0 || indiceProd >= contProductos) return false;  
  
        ProdCarrito prod = Prod[indiceProd];  
        //validemos contra el stock disponible del producto  
  
        if(Ncantidad > prod.producto.getStock()) {  
            JOptionPane.showMessageDialog(null, "Stock insuficiente, solo quedan: " + prod.producto.getStock() + " unidades", "Error", JOptionPane.ERROR_MESSAGE);  
            return false;  
        }  
        prod.cantidad = Ncantidad;  
        return true;  
    }  
  
    //para el total de las compras  
    public static double CalcularTotal () {  
        double total = 0.0;  
        for(int i = 0; i < contProductos; i++) {  
            total += Prod[i].producto.getPrecio() * Prod[i].cantidad;  
        }  
        return total;  
    }  
  
    //para limpiar el carro  
    public static void LimpiarCarro () {  
        //vacuemos el carro reiniciando el arreglo y el contador  
        Prod = new ProdCarrito[MaxProdCarro];  
        contProductos = 0;  
    }  
}
```

Este constructor posee constantes como el máximo número de productos metidos en el carrito de compras, se agregó dos getters para obtener los datos de la cantidad de productos en el carrito y la cantidad de productos como tal, se quedaron métodos para agregar productos al carrito, para eliminar fotos del carrito y actualizar la cantidad de productos en el mismo, se creó un método para calcular el total de la compra y como último un método rápido para limpiar el carro después de realizar a la compra.

Administrador de Usuarios

```
public class AdminDUuarios {  
    //para las sesiones  
  
    private static int sesionesActivas = 0; //contador de sesiones  
  
    private static final String ArchiUser = "Usuarios.csv";  
  
    //una muy bonita matriz para los objetos usuario  
    private static Usuarios[] listaUsuarios = new Usuarios[100]; //numero maximo de usuarios  
    private static int CantUsuarios = 0; //ya se la saben el contrador de usuarios  
  
    //metodos para sincronizar las sesiones de los usuarios  
    public static synchronized void AumentarSesiones () {  
        sesionesActivas++;  
        System.out.println("Sesion iniciada. Activas: " + sesionesActivas); // Mensaje temporal  
    }  
  
    public static synchronized void BajarSesiones () {  
        if (sesionesActivas > 0){  
            sesionesActivas--;  
        }  
        System.out.println("Sesion iniciada. Activas: " + sesionesActivas); // Mensaje temporal  
    }  
  
    //el getter para despues  
    public static synchronized int getSesionesActivas () {  
        return sesionesActivas;  
    }  
}
```

Acá lo que tenemos es el contador de activas la declaración del archivo de usuarios en el formato CSV, tenemos dos métodos uno de ellos está relacionado para sincronizar el inicio de sesión con la cantidad de sesiones activas y el cerrar sesión que se disminuyen, se agrego un get para obtener el dato de sesiones activas.

Carga de Usuarios

```
public static void cargarUsuarios () {
    File archivo = new File(ArchiUsur);
    if (archivo.exists()) {
        JOptionPane.showMessageDialog(null, "El archivo de los usuarios no se encontro, se iniciara como admin", "Error 01", JOptionPane.ERROR_MESSAGE);
        return;
    }
    CANTUSUARIOS = 0; //reiniciamos el contador para cargar
    try (BufferedReader Lector = new BufferedReader(new FileReader(archivo))) {
        String linea;
        while ((linea = Lector.readLine()) != null && CANTUSUARIOS < listaUsuarios.length) {
            String[] datos = linea.split(",");
            //comprobamos para que haya diferencia entre vendedores y los demás usuarios
            if (datos.length > Usuarios.CAMPOS) {
                String tipo = datos[Usuarios.TIPOUSUR].trim();
                listaUsuarios[CANTUSUARIOS++] = new Usuarios(
                    datos[Usuarios.ID].trim(),
                    datos[Usuarios.NOMBRE].trim(),
                    datos[Usuarios.CONTRASEÑA].trim(),
                    datos[Usuarios.TIPOUSUR].trim()
                );
            }
        }
        if (CANTUSUARIOS > 0) {
            // JOptionPane.showMessageDialog(null, "Los usuarios creados se han cargado", "Acción exitosamente exitosa", JOptionPane.INFORMATION_MESSAGE);
        }
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Algo salio mal en la carga de los archivos", "Error 03", JOptionPane.ERROR_MESSAGE);
    }
}
```

En este método se hace llamado al archivo de usuarios haciendo lectura del mismo y de igual manera con un bucle que nos permite cargar todos los usuarios creados al archivo SV con los campos de código, nombre, contraseña, tipo de usuario.

Guardar Usuarios

```
public static void GuardarUsuarios () {
    try (PrintWriter escribir = new PrintWriter(new FileWriter(ArchiUsur))) {
        for (int i = 0; i < CANTUSUARIOS; i++) {
            Usuarios u = listaUsuarios[i];
            String lineaCSV = u.getId() + "," + u.getNombre() + "," + u.getContraseña() + "," + u.getTipoUsuario();
            escribir.println(lineaCSV);
        }
        JOptionPane.showMessageDialog(null, "Los usuarios fueron guardados en el archivo correspondiente", "Acción Exitosa", JOptionPane.INFORMATION_MESSAGE);
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Algo salio mal al guardar a los usuarios", "Error 04", JOptionPane.ERROR_MESSAGE);
    }
}
//un metodo para validar las credenciales de los usuarios
```

Éste método nos permite escribir sobre el archivo CSV recorriendo la cantidad de usuarios creada escribiendo sobre el archivo CSV con el ID, el nombre, la contraseña y el tipo de usuario

Autenticacion de Usuarios

```
public static Usuarios autenticacion (String id, String Contraseña) {
    for (int i = 0; i < CANTUSUARIOS; i++) {
        Usuarios u = listaUsuarios[i];
        //asumimos que el usuario y contraseña existen
        if (u.getId().equals(id) && u.getContraseña().equals(Contraseña)) {
            return u;
        }
    }
    return null; //la operacion salio mal xd
}
```

Este método nos permite recorrer la matriz de usuarios y la lista de los mismos para encontrar si la contraseña ingresada corresponde a la creada igual que el código del usuario.

Agregar Usuarios

```
public static void AgregarUsuario (Usuarios nUsuarios) {
    if (CantUsuarios < listadUsuarios.length){
        listadUsuarios [CantUsuarios ++] = nUsuarios;
        GuardarUsuarios ();
    } else {
        JOptionPane.showMessageDialog (null, "se ha llegado al maximo de usuarios registrados", "Error 05", JOptionPane.WARNING_MESSAGE);
    }
}
```

Este método nos permite agregar a la lista y cantidad de usuarios los usuarios que han sido creados independientemente sean vendedores o clientes.

Código Repetido

```
//un metodo auxiliar para ver si es unico el codito calro
public static boolean CodRepetido (String id) {
    for (int i = 0; i < CantUsuarios ; i++) { //buscar en la matriz de usuarios
        if (listadUsuarios [i].getId().equals(id)) {
            return true;
        }
    }
    return false;
}
```

Este método nos permite averiguar si el código de algún usuario está repetido en la matriz y lista de usuarios

Actualizar Usuarios

```
public static boolean ActualizarUsuario (String id, String Nnombre, String Ncontraseña) {
    for (int i = 0; i < CantUsuarios ; i++) {
        if (listadUsuarios [i].getId().equalsIgnoreCase(id)){
            listadUsuarios [i].nombre = Nnombre;
            listadUsuarios [i].Contraseña = Ncontraseña;
            return true;
        }
    }
    return false;
}
```

Este método nos permite actualizar lo que es el nombre y la contraseña del usuario.

Eliminar usuarios

```
public static boolean EliminarUsuario (String Id) {  
    for (int i = 0; i < CantUsuarios ; i++){  
        if (listadUsuarios [i].getId().equalsIgnoreCase(Id)){  
            //Desplazamos todos los usuarios para llenar el espacio  
            for (int j = i; j < CantUsuarios - 1; j++){  
                listadUsuarios [j] = listadUsuarios [j + 1];  
            }  
            listadUsuarios [CantUsuarios - 1] = null; //limpiamos la ultima linea  
            CantUsuarios --;  
            return true;  
        }  
    }  
    return false;  
}  
}  
//los metodos de vendedor los quite de aca y los puse en su lugar en administracion de vendedores xd,
```

Este método de igual manera recorre la matriz de usuarios buscando el código del usuario en cuestión para luego permitir eliminarlo de la matriz de usuarios.

Administrador de Vendedores

```
public class AdminDVendedores {  
  
    private static final String Archivo_Vendedor = "Vendedores.csv";  
  
    private static final String Estado_Vendedor = "vendedores.ser"; //ser  
  
    private static final int MVendedores = 50;  
    private static final int NUM_CAMPOS_VENDEDOR = 7; // Ahora son 7 columnas por lo del reporte y las weas  
    //esta es la matriz solo para los vendedores  
    private static Vendedor[] listadVendedores = new Vendedor[MVendedores];  
    private static int CantVendedores = 0;
```

Para la clase de administrador de vendedores se declararon las constantes que corresponden al archivo CSV de vendedores el archivo .ser de los mismos se declaró la cantidad máxima de vendedores se declaró la matriz de vendedores y el campo numérico que corresponde.

Cargar Vendedores

```
public static void CargarVendedores () {
    File archivo = new File("Archivo_Vendedor");
    if (!archivo.exists()) {
        JOptionPane.showMessageDialog(null, "El archivo de vendedores no fue encontrado.", "Error 015", JOptionPane.ERROR_MESSAGE);
        return;
    }
    ContVendedores = 0;
    try (BufferedReader Lector = new BufferedReader(new FileReader(archivo))) {
        String linea;
        int num = 0;
        Lector.readLine();
        while ((linea = Lector.readLine()) != null && ContVendedores < listaVendedores.length) {
            String[] datos = linea.split(",");
            //patron vendedor: Codigo, nombre, genero, contraseña, ventas
            if (datos.length > NUM_CAMPOS_VENDEDOR) {
                int ventas = 0;
                try {
                    String id = datos[0].trim();
                    String nombre = datos[1].trim();
                    String Contraseña = datos[2].trim();
                    //el tipo de usuario lo guardado en datos[3]
                    String genero = datos[4].trim();
                    ventas = Integer.parseInt(datos[5].trim());
                    double totalVentas = Double.parseDouble(datos[6].trim());
                } catch (NumberFormatException nfe) {
                    System.out.println("Advertencia: Total de ventas invalido para vendedor " + id + ". Usando 0.0.");
                }
            }
            if (datos.length > 5 && (datos[5].trim().isEmpty())) {
                try {
                    totalVentas = Double.parseDouble(datos[5].trim());
                } catch (NumberFormatException nfe) {
                    System.out.println("Advertencia: Total de ventas invalido para vendedor " + id + ". Usando 0.0.");
                }
            }
            Vendedor Nvendedor = new Vendedor(id, nombre, Contraseña, genero);
            Nvendedor.setTotalVentasHechas(ventas);
            Nvendedor.setTotalVentasHechas(totalVentas);
            listaVendedores[ContVendedores] = Nvendedor;
            if (!AdminUsuarios.CodRepetido(id)) { //asi evitamos los duplicados si en AdminUsuarios ya lo cargo
                AdminUsuarios.AgregarUsuario(Nvendedor);
            }
        }
    } catch (FileNotFoundException e) {
        JOptionPane.showMessageDialog(null, "Error al cargar el archivo CSV de vendedores.", "Error 17", JOptionPane.ERROR_MESSAGE);
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Error al crear el vendedor desde el CSV" + linea, "Error 16", JOptionPane.ERROR_MESSAGE);
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, "Error al cargar el archivo CSV de vendedores.", "Error 17", JOptionPane.ERROR_MESSAGE);
    }
}
```

Este método se encarga de por medio del archivo SVD vendedores cargar todos los vendedores creados en el archivo CSV al programa para ser más específicos, cargarlos a la matriz de vendedores, esto por medio de un bucle que se repite indefinidamente por la cantidad de vendedores que estén en el archivo, al cargarse se van a cargar en el orden de código, nombre, contraseña, tipo de usuario, género, ventas, total de ventas, al finalizar este bucle se hace una comprobación del total de las ventas, al terminar el bucle se crea el vendedor dentro de la matriz de vendedores para asegurarse de qué se encuentren en esta, aumentando la cantidad de vendedores, se hace una pequeña comprobación sobre el código repetido para evitar duplicaciones.

Cargar Vendedores

```
public static void GuardarVendedor () {
    try (PrintWriter escribir = new PrintWriter(new FileWriter("Archivo_Vendedor"))){
        escribir.print("codigo,nombre,contraseña,tipoUsuario,genero,ventasHechas,totalVentasHechas");
        for (int i = 0; i < ContVendedores; i++) {
            Vendedor v = listaVendedores[i];
            if (v != null) {
                //Formato del archivo, cod, nombre, genero, contraseña y ventas y si el mp haber agregadp el tipo de usuario no me dejaba iniciar sesion
                String lineaCSV
                    = v.getId() + ","
                    + v.getNombre() + ","
                    + v.getContraseña() + ","
                    + v.getTipoUsuario() + ","
                    +//ESTA MALDITA LINEA ME KGO LA NOCHE :/
                    v.getGenero() + ","
                    + v.getTotalVentasHechas() + ","
                    + v.getTotalVentasHechas(); //las ventas hechas osea todasa pues
                escribir.println(lineaCSV);
            }
        }
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Error al guardar vendedores en CSV.", "Error 18", JOptionPane.ERROR_MESSAGE);
    }
    Guardarestado();
}
```

Este método se encarga de guardar en el archivo de vendedores, los vendedores creados esto recorriendo un ciclo que inicia desde cero hasta la cantidad de vendedores creados en el archivo SV se escribe de la siguiente manera: código, el nombre, contraseña, tipo de usuario, género, total de ventas y venta realizadas.

Código Repetido

```
public static boolean CodRepetido (String id) {  
    for (int i = 0; i < CantVendedores ; i++) { //buscar en la matriz de vendedore  
        if (listadVendedores [i].getId().equalsIgnoreCase(id)) {  
            return true;  
        }  
    }  
    return false;  
}
```

Este método se encarga de recorrer desde cero hasta la matriz de vendedores, buscando el código del vendedor para evitar duplicados.

Creación de Vendedor

```
//creamos y agregamos un nuevo vendedor si el código SI ES unico  
public static boolean CreacionVendedor (String id, String nombre, String genero, String contrseña) {  
    if (CantVendedores >= Mvendedores ) {  
        JOptionPane.showMessageDialog (null, "El límite de vendedores fue alcanzado", "Error 06", JOptionPane.ERROR_MESSAGE);  
        Bitacora.RegistrarEvento (Bitacora.Tipo_Admin, "Admin", Bitacora.OP_Crear_Vendedor , Bitacora.ESTADO_FALLIDA , "Maximo de vendedores alcanzado");  
  
        return false;  
    }  
    if (CodRepetido (id)) {  
        JOptionPane.showMessageDialog (null, "El código de vendedor ya está en uso", "Error 07", JOptionPane.ERROR_MESSAGE);  
        Bitacora.RegistrarEvento (Bitacora.Tipo_Admin, "Admin", Bitacora.OP_Crear_Vendedor , Bitacora.ESTADO_FALLIDA , "Codigo repetido");  
  
        return false;  
    }  
    Vendedor Nvendedor = new Vendedor (id, nombre, contrseña, genero);  
    listadVendedores [CantVendedores ++] = Nvendedor;  
    GuardarVendedor ();  
    AdminUsuarios.AgregarUsuario (Nvendedor);  
    Bitacora.RegistrarEvento (Bitacora.Tipo_Admin, "Admin", Bitacora.OP_Crear_Vendedor , Bitacora.ESTADO_EXITOSA , "Creacion de vendedor exitosa");  
    return true;  
}
```

Este método se encarga de llamar al constructor de vendedores para esta manera crear un nuevo vendedor y guardarlo en la matriz de vendedores haciendo llamado también al método de código repetido para evitar duplicados, y por último se hace un llamado al método de agregar usuario para agregarlo al archivo CSV de usuarios.

Modificación de Vendedor

```
public static boolean ModVendedor (String id, String Nnombre, String Ncontraseña) {  
    Vendedor v = BuscarVendedor (id);  
    if (v != null) {  
        v.setNombre (Nnombre);  
        v.setContraseña (Ncontraseña);  
        GuardarVendedor ();  
        Bitacora.RegistrarEvento (Bitacora.Tipo_Admin, "Admin", Bitacora.OP_Mod_Vendedor , Bitacora.ESTADO_EXITOSA , "Modificacion de vendedor exitosa");  
  
        return true;  
    }  
    return false;  
}
```

Este método se encarga de llamar al método de búsqueda vendedor, al encontrar el vendedor hacemos una carga al, set, equivalente del nombre y contraseña para de esta manera poder modificar estos atributos del vendedor seguido se hace en llamado a guardar vendedor para guardar los cambios en la matriz de vendedores y por lo consiguiente en el archivo CSV de usuarios.

Eliminacion de Vendedor

```
//un metodo de eliminacion por el codigo
public static boolean EliminarVendedor (String id) {
    for (int i = 0; i < CantVendedores ; i++) {
        if (listadVendedores [i].getId().equals(id)) {
            //el todo confiable ciclo for
            for (int j = i; j < CantVendedores - 1; j++) {
                listadVendedores [j] = listadVendedores [j + 1];
            }
            listadVendedores [CantVendedores - 1] = null; //eliminamos la ultima referencia
            CantVendedores--;
            GuardarVendedor ();
            Bitacora.RegistrarEvento (Bitacora.Tipo_Admin, "Admin", Bitacora.OP_Eli_Vendedor, Bitacora.ESTADO_EXITOSA, "Eliminacion de vendedor exitosa");
        }
        return true;
    }
    return false;
}
```

Éste método se encarga de buscar en la matriz de vendedores, por el código de vendedor para esta manera eliminar el vendedor de la materia de vendedores por consiguiente también guardando este cambio en una materia de usuarios.

Validacion de Vendedor

```
public static Vendedor ValidIngreso (String id, String Contraseña) {
    //buscamos al vendedor por codigo
    Vendedor v = BuscarVendedor (id);
    //si lo encuentra y la contraseña existe
    if (v != null && v.getContraseña().equals(Contraseña)) {
        return v;
    }
    return null;
}
```

Éste método se encarga de validar la contraseña del vendedor para esta manera poder ingresar como vendedor al programa.

Datos para la tabla de Vendedor

```
//para la tabla de vendedores LPM
public static Object[][] DatosTablaVendedor () {
    //miramos si hay vendedores cargados para evitar errores
    if (listadVendedores == null || CantVendedores == 0){
        return new Object[0][4];
    }
    Object[][] datos = new Object[CantVendedores ][4];
    for (int i = 0; i < CantVendedores ; i++) {
        //obtenemos el objeto vendedor de la lista
        Vendedor v = listadVendedores [i];
        datos[i][0] = v.getId();
        datos[i][1] = v.getNombre();
        datos[i][2] = v.getGenero();
        //ESTO IMPEDIA QUE LA MALDITA TABLA SE ACTUALICE
        datos[i][3] = v.getVentasHechas();
    }
    return datos;
}
```

En este método se encarga de obtener los datos necesarios para que en la tabla del menú de vendedores se actualice y esté conforme a el código, nombre, género, total de ventas.

Get Lista de Vendedores

```
//para el fokin reporte
public static Vendedor[] getListadVendedores () {
    //creamos una copia para evitar modificaciones externas por si las moscas
    Vendedor[] copia = new Vendedor[CantVendedores];
    System.arraycopy(listadVendedores , 0, copia, 0, CantVendedores);
    return copia;
}

public static int getCantVendedores () {
    return CantVendedores;
}
```

Éste método más que nada se encarga de crear una copia de la matriz de vendedores para, en el módulo de reportes poder usar esa matriz.

Administrador de Productos

```
/*
public class AdminDProductos {

    private static final String ArchiProd = "Productos.csv"; //csv
    private static final String Estado_Producto = "Productos.ser"; //ser

    private static final int MProductos = 100; //numero maximo de productos aunque... realmente, para este proyecto es demasiado, igual que esta desc
    private static final String ArchiStock = "Stock.csv";
    //la matriz para los productos tecnologicos como esta macbook air de 256gb con el chip m1 y la comida como este sabrosa tortrix de limon ☺
    private static Productos[] listaProductos = new Productos[MProductos];
    private static int CantidadProducto = 0;
}
```

En este método y clase contiene todo lo que está relacionado con los productos inicialmente se tiene las constantes que corresponden a el archivo CSV de productos y el archivo .ser de los mismos se crearon las constantes y la materia para los productos y la cantidad de los mismos.

Cargar Productos

```
public static void CargarProductos () {
    File Archivo = new File(ArchiProd);
    if (Archivo.exists()) {
        JOptionPane.showMessageDialog(null, "El archivo de productos, no se encontro, pero aun asi lo incluimos vacio", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }
    CantidadProducto = 0;
    try (BufferedReader lector = new BufferedReader(new FileReader(Archivo))) {
        String Linea;
        while ((Linea = lector.readLine()) != null && CantidadProducto < listaProductos.length) {
            String[] datos = Linea.split(",");
            if (datos.length >= 6) {
                try {
                    String codigo = datos[0].trim();
                    String nombre = datos[1].trim();
                    String categoria = datos[2].trim();
                    double precio = Double.parseDouble(datos[3].trim());
                    int stock = Integer.parseInt(datos[4].trim());
                    Productos Nproducto = null;
                    if (categoria.equals("TECNOLOGIA")) {
                        //cargamos los datos Ya se lo saben, no quiero explicar mas
                        int garantia = Integer.parseInt(datos[5].trim()); //el indice directo
                        Nproducto = new Productos(codigo, nombre, categoria, precio, stock, garantia);
                    } else if (categoria.equals("ALIMENTO")) {
                        //cargamos los datos del producto de comida
                        String caducidad = datos[5].trim();
                        Nproducto = new ProductosComida(codigo, nombre, precio, stock, caducidad);
                    } else if (categoria.equals("OTROS")) {
                        Nproducto = new ProductosOtros(codigo, nombre, precio, stock);
                    } else {
                        JOptionPane.showMessageDialog(null, "Error el formato de linea o la categoria no es la correcta", "Error", JOptionPane.ERROR_MESSAGE);
                        continue;
                    }
                    listaProductos[CantidadProducto] = Nproducto;
                    CantidadProducto++;
                } catch (Exception e) {
                    JOptionPane.showMessageDialog(null, "Error al procesar una linea en Productos.csv.\nLinea con problemas: " + Linea + "\nError: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
                } catch (IOException e) {
                    JOptionPane.showMessageDialog(null, "Error al cargar el archivo CSV de productos.", "Error", JOptionPane.ERROR_MESSAGE);
                }
            }
        }
    }
}
```

Igual que en el método de cargar vendedores hacemos llamado para el archivo de productos recorriendo la matriz y la cantidad de los mismos para de esta manera escribir archivo CSV, con los datos correspondientes a los productos que son: código, nombre, categoría, precio, Stock, Y en parte de categoría se tiene tres condiciones siendo caso el producto es de tecnología es un elemento o no pertenece a ningún estas dos clases añadiendo como atributo específico, los meses de garantía para

puntos de tecnología, la fecha de caducidad para los productos alimenticios y nada para los productos que no concuerden a ninguna de estas categorías.

Guardar Productos

```
public static void GuardarProductos () {
    try (PrintWriter escribir = new PrintWriter(new FileWriter(ArchiProd))) {
        for (int i = 0; i < CantProducto; i++) {
            Productos p = listadProductos [i];
            //usamos el tocsvline para obtener linea completa
            escribir.println(p.toCsLine());
        }
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Error al guardar los productos en el csv", "Error", JOptionPane.ERROR_MESSAGE);
    }
    Guardarestado ();
}
```

Éste método se encarga de recorrer la matriz de vendedores para guardar en el archivo CSV los vendedores que han sido creados.

Buscar Productos y código repetido

```
public static boolean CodRepetido (String codigo) {
    for (int i = 0; i < CantProducto; i++) {
        if (listadProductos [i].getCodigo().equalsIgnoreCase(codigo)) {
            return true;
        }
    }
    return false;
}

public static Productos BuscarProd (String codigo) {
    //buscamos los productos en la matriz de productos
    for (int i = 0; i < CantProducto; i++) {
        String codigoEnLista = listadProductos [i].getCodigo();

        if (codigoEnLista.equalsIgnoreCase(codigo)) {
            return listadProductos [i];
        }
    }
    return null;
}
```

Éstos métodos se encargan de buscar en la matriz de productos por el código del producto para evitar un duplicado, mientras que el otro se encarga de buscar en la matriz de productos buscando el código del producto que volviendo el producto encontrado.

Creación de productos

```
public static boolean CreadorProducto(String codigo, String nombre, double precio, int stock, String categoria, String Atributo){  
    if (Contenedor > 10000) {  
        JOptionPane.showMessageDialog(null, "Se llego al limite de productos", "Error", JOptionPane.ERROR_MESSAGE);  
        Bitacora.RegistrarEvento(Bitacora.Tipo_Admin, "admin", Bitacora.OP_Crear_Producto, Bitacora.ESTADO_FALLIDA, "Limite de productos alcanzado");  
        return false;  
    }  
    if (CodigoFalso(codigo)) {  
        JOptionPane.showMessageDialog(null, "Ese codigo ya esta en uso", "Error", JOptionPane.ERROR_MESSAGE);  
        Bitacora.RegistrarEvento(Bitacora.Tipo_Admin, "admin", Bitacora.OP_Crear_Producto, Bitacora.ESTADO_FALLIDA, "Codigo de producto repetido");  
        return false;  
    }  
    Productos Nproducto = null;  
  
    String catrga = categoria.toUpperCase();  
    try{  
        if (catrga.equals("TECNOLOGIA")){  
            int garantia = Integer.parseInt(Atributo);  
            Nproducto = new ProductoTec(codigo, nombre, precio, stock, garantia);  
        } else if (catrga.equals("ALIMENTO")){  
            //la fecha de vencimiento ya viene como string  
            Nproducto = new ProductoComida(codigo, nombre, precio, stock, Atributo);  
        } else if (catrga.equals("OTROS")){  
            Nproducto = new ProductoOtros(codigo, nombre, precio, stock);  
        } else {  
            JOptionPane.showMessageDialog(null, "Categoria de producto no valida", "Error", JOptionPane.ERROR_MESSAGE);  
            return false;  
        }  
    } catch (NumberFormatException e){  
        JOptionPane.showMessageDialog(null, "El atributo especifico no es valido para la categoria", "Error", JOptionPane.ERROR_MESSAGE);  
        Bitacora.RegistrarEvento(Bitacora.Tipo_Admin, "admin", Bitacora.OP_Crear_Producto, Bitacora.ESTADO_FALLIDA, "Creation de producto fallida");  
        return false;  
    }  
    if (Nproducto != null){  
        listaProductos[ContProducto] = Nproducto;  
        GuardarProductos();  
        Bitacora.RegistrarEvento(Bitacora.Tipo_Admin, "admin", Bitacora.OP_Crear_Producto, Bitacora.ESTADO_EXITOSA, "Creation de producto exitosa");  
        return true;  
    }  
    return false;  
}
```

En este método nos encargamos de hacer llamado al constructor de productos primera una validación si la cantidad de productos creados es mayor a la permitida siguiendo con la validación si el código del producto está repetido, por consiguiente se llega a la parte de la categoría que con simples condiciones si cumple con condición de ser de tecnología el producto presentará la garantía si es un alimento presentará la fecha de caducidad y sin pertenece a la categoría de otros no presenta ninguna tributo único al final se guardará esto en la matriz de productos.

Modificación de productos

```
//los demas metodos faltan todavia xd  
public static boolean ModProductos (String Codigo, String Nombre, String Natributo) {  
    Productos p = BuscarProd (Codigo);  
    if (p != null){  
        p.setNombre(Nombre);  
        p.setStock(MProductos);  
        return true;  
    }  
    try{  
        if (p instanceof productoTec){  
            productoTec Pt = (productoTec) p;  
            int garantia = Integer.parseInt (Natributo);  
            Pt.setMesesGarantia(garantia);  
        } else if (p instanceof productoComida){  
            productoComida Pa = (productoComida) p;  
            Pa.setFechaVencer(Natributo);  
        } else {  
            //por si las moscas  
            JOptionPane.showMessageDialog(null, "Tipo de producto desconocido", "Error", JOptionPane.ERROR_MESSAGE);  
            return false;  
        }  
    } catch (NumberFormatException e){  
        JOptionPane.showMessageDialog(null, "el nuevo valor especifico (La garantia o la caducidad) no es valido", "Error", JOptionPane.ERROR_MESSAGE);  
    }  
    GuardarProductos();  
    return false;  
}
```

En este método hemos llamado a el constructor de productos y buscamos el producto por su código, de esta manera podemos hacer llamado para cambiarle el nombre y la cantidad disponible entre los productos, en el momento de hacer una modificación de atributos específico se tienen instancias si el producto es de tecnología o alimento, de esta manera para poder modificar su tributo específico al finalizarse guardará estos cambios en la materia de productos.

Eliminación de productos

```
//un metodo de eliminacion por el codigo
public static boolean EliminarProducto (String codigo) {
    for (int i = 0; i < CantProducto ;i++){
        if (listadProductos [i].getCodigo().equals(codigo)) {
            //el todo confiable ciclo for
            for (int j = i; j < CantProducto - 1; j++) {
                listadProductos [j] = listadProductos [j + 1];
            }
            listadProductos [CantProducto - 1] = null; //eliminamos la ultima referencia
            CantProducto--;
            GuardarProductos ();
            Bitacora.RegistrarEvento (Bitacora.Tipo_Admin , "admin" , Bitacora.OP_ELI_Producto , Bitacora.ESTADO_EXITOSA , "Eliminacion de producto exitosa");
            return true;
        }
    }
    return false;
}
```

En este método recorrimos la matriz de productos para encontrar el producto con el código que estamos buscando para esta manera poder eliminar al producto y luego guardaré dos cambios en la matriz de productos.

Carga de productos

```
public static String CargaProductos (String Archiv) {
    int producCargado = 0;
    int lineaFail = 0;
    StringBuilder Error = new StringBuilder();
    try (BufferedReader lector = new BufferedReader(new FileReader(Archiv))) {
        String linea;
        int numLinea;
        while ((linea = lector.readLine()) != null) {
            numLinea++;
            String lineaT = linea.trim();
            if (linea.isEmpty()) {
                continue;
            }
            String[] datos = linea.split(",");
            //validamos que haya al menos 6 campos los 5 normales + el atributo unico
            if (datos.length != Productos.CAMPOS_Prod) {
                Error.append("Linea").append(numLinea).append(": formato incorrecto, se esperan 6 lineas").append(datos.length).append("\n");
                lineaFail++;
                continue;
            }
            try {
                //mapreamos los datos
                String codigo = datos[0].trim();
                String nombre = datos[1].trim();
                String precioStr = datos[2].trim();
                String stockStr = datos[3].trim();
                String categoria = datos[4].trim().toUpperCase();
                String atributo = datos[5].trim();
                //paramos los numeros
                double precio = Double.parseDouble(precioStr);
                int stock = Integer.parseInt(stockStr);
                }
            catch (Exception e) {
                //capturaremos error en el precio o stock o las validaciones de gestion de productos
                Error.append("Linea").append(numLinea).append(": error de datos").append(e.getMessage()).append("\n");
                lineaFail++;
            }
        }
        if (producCargado > 0) {
            GuardarProductos ();
        }
    } catch (FileNotFoundException e) {
        return "Error al no encontrar al archivo ";
    } catch (IOException e) {
        return "Error al leer el archivo " + e.getMessage();
    }
    if (lineaFail > 0) {
        Error.insert(0, "resumen: " + producCargado + ": productos cargados " + lineaFail + " lineas fallaron \n");
        return Error.toString();
    } else {
        Bitacora.RegistrarEvento (Bitacora.Tipo_Admin , "admin" , Bitacora.OP_Cargar_Producto , Bitacora.ESTADO_EXITOSA , "Carga de productos exitosa");
        return "carga de archivos exitosa :D " + producCargado + " productos creados ";
    }
}
```

```
//validaciones de los valores de precio y stock
if (precio < 0 || stock < 0) {
    throw new Exception("El precio y el stock deben ser valores positivos.");
}
//previsualizar el atributo y el formato
if (categoria.equals("TECNOLOGIA")){
    try{
        int garantia = Integer.parseInt(atributo);
        if (garantia < 0) {
            throw new Exception("La garantia debe de ser un numero entero");
        }
    } catch (NumberFormatException e) {
        throw new Exception("La garantia no es un valor valido");
    }
} else if (categoria.equals("OTROS")){
    //per si los moscas, aunque la categoria otros ignora el atributo especifico xd, pero en fin, para que esta mrd no se caiga a pedazos
    atributo = "NA"; //el atributo
} else if (categoria.equals("ALIMENTO")){
    if (productosComida.FechaValida(atributo)){
        throw new Exception("Formato de fecha no valido, formato esperado(YYYY-MM-DD)");
    }
}
//llamamos el metodo de crear producto, para el manejo de errores como el codigo repetido, y el formato de fecha y garantia
if (CreacionProducto (codigo, nombre, precio, stock, categoria, atributo)){ 
    producCargado++;
}
else {
    //per si falla debido a salir el mensaje de las validaciones pero por si los moscas
    Error.append("Linea").append(numLinea).append(": fallo al crearse, posiblemente, por el codigo duplicado, el precio o stock en invalidos o el limite de productos ya se alcanzo \n");
    lineaFail++;
}
} catch (Exception e) {
    //per si falla en el precio o stock o las validaciones de gestion de productos
    Error.append("Linea").append(numLinea).append(": error de datos").append(e.getMessage()).append("\n");
    lineaFail++;
}
}
```

En este método se hace una carga de los datos del archivo CSV al programa y por consiguiente en la matriz de productos esto lo hacemos los datos conforme a lo que corresponde el dato en la matriz de productos de esta manera los tenemos creados, por lo que se sigue con las validaciones en lo que corresponde tecnología que la garantía no sea menor que es cero y que sea un numero y que el

alimento la fecha de caducidad sea la correcta al salir todo bien se crea los productos de nuevo en la matriz por consiguiente guardándolos también en la misma, como último se tienen resumen de productos cargados y fallos en el mismo.

Datos de tabla de productos

```
//para la tabla
public static Object[][] DatosTablaProd() {
    //las 3 columnas codigo, nombre y la categoria
    if (listadProductos == null || CantProducto == 0) {
        return new Object[0][5];
    }
    Object[][] datos = new Object[CantProducto ][5];

    for (int i = 0; i < CantProducto ; i++) {
        Productos p = listadProductos [i];

        datos[i][0] = p.getCodigo();
        datos[i][1] = p.getNombre();
        datos[i][2] = p.getCategoría();
        datos[i][3] = p.getPrecio();
        datos[i][4] = p.getStock();

    }
    return datos;
}
```

En este método recorremos la matriz y lista de productos para obtener los datos que corresponden al código, nombre, categoría, precio, Stock para la tabla del menú de administrador y por consiguiente de los demás menús también.

Agregar Stock

```
//para agregar el stock
public static boolean agregarStock (String IdProd, int Cant, String usuarioId, String nombreUsur) {
    //buscamos el producto por su codigo
    Productos p = BuscarProd (IdProd);
    if (p != null) {
        //obtenemos el stock actual y sumar lo que vamos a meter
        int NStock = p.getStock() + Cant;
        //actualizamos el objeto en la memoria
        p.setStock(NStock);

        //hacemos nuestro llamado para que se guarde
        RegistrarStock (IdProd, Cant, usuarioId, nombreUsur);
        GuardarProductos ();
        Bitacora.RegistrarEvento (Bitacora.Tipo_Vendedor, usuarioId, Bitacora.OP_Agregar_Stock , Bitacora.ESTADO_EXITOSA , "Se agrego stock a los productos");
        return true;
    }
    return false;
}
```

En este método buscamos código al producto, luego declaramos y llevamos al Stock actual del producto y le creamos la cantidad que los usuarios desea agregar al terminar guardamos ese stock nuevo para luego guardarlo en la matriz de productos otra vez.

Registrar Stock

```
public static void RegistrarStock (String codigo, int Cantidad, String usuarioId, String nombreUser) {
    File ArchivoHis = new File("HistorialStock.csv");
    boolean Narchivo = ArchivoHis.exists();
    try (PrintWriter esribit = new PrintWriter(new FileWriter(ArchivoHis, true))) {
        //obtener la fecha
        if (Narchivo) {
            esribit.println("fecha_hora_iso,codigo_producto,cantidad_agregada,usuario_id,usuario_nombre");
        }
        LocalDateTime ahora = LocalDateTime.now();
        DateTimeFormatter formato = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        String horafecha = ahora.toString();
        //creamos la linea para guardar
        String lineacsv = horafecha + "," + codigo + "," + Cantidad + "," + usuarioId + "," + nombreUser;
        esribit.println(lineacsv);
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "No se pudo registrar el movimiento en el historial de stock.", "Error de Historial", JOptionPane.ERROR_MESSAGE);
    }
}
```

Se hace llamado al archivo de historial de Stock para luego escribir sobre él de esta manera escribimos lo que es la fecha y la hora en formato ISO, el código del producto, la cantidad agragada por el usuario, el código del usuario y el nombre del usuario de esta manera hacemos llamado a los librerías para la hora y el formato Dora para guardar esto último se crea una línea para guardar todos los datos anteriormente mencionados.

Datos de la tabla de cliente y reservar de Stock

```
//para la tabla del cliente xd
public static Object[][] DatosTablaCliente () {
    //creamos un arreglo de 4 columnas, codigo, nombre categoria y el stock disponible
    Object[][] datos = new Object[CantProducto] [4];
    for (int i = 0; i < CantProducto; i++) {
        Productos p = listadProductos[i];
        //los datos de la tabla xd
        datos[i][0] = p.getCodigo();
        datos[i][1] = p.getNombre();
        datos[i][2] = p.getCategoría();
        datos[i][3] = p.getStock();
    }
    return datos;
}

//Para el wn del vendedor xd
public static void ReservaStock (String codigoProducto, int CantComprar) {
    Productos p = BuscarProd(codigoProducto);
    if (p != null) {
        p.setStock(p.getStock() - CantComprar);
    }
}
```

Éste primer método se encarga de hacer llamado a los datos de la matriz de productos para luego hacer llamado para obtener los datos de: código, nombre, categoría, Stock para la tabla que corresponde al menú de cliente, en los 10 métodos se encarga de buscar en la materia de productos por el producto que queremos para al final de la compra restar la cantidad te compra con el Stock actual.

Administrador de Pedidos

```
public class AdminDPedidos {  
  
    //archivo y el maximo de pedidos  
    private static final String ArchivPedidos = "Pedidos.csv";  
    private static final String ESTADO_PEDIDOS_SER = "PedidosPendientes.ser";  
    private static final int MaxPedidos = 200;  
  
    private static Pedidos[] listadPedidos = new Pedidos[MaxPedidos];  
    private static int CantPedidos = 0;  
  
    //el get para la chingadera de los hilos  
    public static int getCantPedidosPendientes () {  
        return CantPedidos; //devolvemos el contador de pedidos sin confirmar  
    }  
}
```

En esta clase se declararon las constancias respectivas del archivo para pedidos y pedidos pendientes, en terminación CSV y .ser, se declararon la matriz de la cantidad y el máximo de pedidos a poder crear.

Datos de la tabla

```
public static Object[][][] DatosTabla () {  
  
    Object[][][] datos = new Object[CantPedidos][][];  
    DateTimeFormatter formato = DateTimeFormatter.ofPattern ("yyyy-MM-dd HH:mm:ss");  
  
    for (int i = 0; i < CantPedidos; i++) {  
        Pedidos p = listadPedidos [i];  
  
        datos[i][0] = p.getIdPedido();  
        datos[i][1] = p.getFechaGeneracion().format(formato);  
        datos[i][2] = p.getIdCliente();  
        datos[i][3] = p.getNombreCliente();  
        datos[i][4] = p.getTotal();  
    }  
    return datos;  
}
```

En este método hacemos uso de un objeto para contener los cinco datos correspondientes al pedido que es: código de pedido, fecha de generación, código del cliente, nombre del cliente y el total.

Crear pedido

```
public static boolean CrearPedido (Cliente cliente) {
    if (Carrito.getCartProd () == 0) {
        JOptionPane.showMessageDialog (null, "Tu carrito se encuentra vacio", "Error", JOptionPane.WARNING_MESSAGE);
        return false;
    }
    //Generamos un id unico para el peido, lo haremos lo mas simple posible no me quiero complicar la vida xd
    String NIDPedido = "PE-" + (CantPedidos + 101);
    double total = Carrito.CalcularTotal ();
    ProdCarrito[] ProdsComprados = new ProdCarrito[Carrito.getCartProd ()];
    System.arraycopy (Carrito.getProds (), 0, ProdsComprados, 0, Carrito.getCartProd ());
    Pedidos Npedido = new Pedidos (NIDPedido, LocalDateTime.now (), cliente.getId (), cliente.getNombre (), total, ProdsComprados, Carrito.getCartProd ());
    //por aca asicreamos la lista de productos del carro con el pedido
    listaPedidos [CantPedidos ++] = Npedido;
    Carrito.LimpiarCarro ();
    guardarEstado ();
    //guardamos el cambio de stock en el archivo de productos
    JOptionPane.showMessageDialog (null, "El pedido se ha realizado", "Pedido Creado", JOptionPane.INFORMATION_MESSAGE);
    Bitacora.RegistrarEvento (Bitacora.OP_El_Cliente, cliente.getId (), Bitacora.OP_Realizar_Pedido, Bitacora.ESTADO_EXITOSA, "Pedido realizado");
    return true;
}
```

Este método se encarga de crear los pedidos opciones de la validación si el pura cantidad de productos es igual a cero nos mandaron un mensaje, siguiente queremos un código único para cada pedido y declaramos el valor del total de la compra, luego hacemos llamado al producto del carrito y hacemos llamado a la matriz de productos de esta manera quedamos nuevo pedido, se guarda en la lista de pedidos y luego se limpia el carrito de compras haciendo el llamado a los respectivo método.

Confirmación de pedido

```
public static boolean ConfirmarPedido (String idPedido, Vendedor vendedor) {
    int indPedido = -1;
    Pedidos PedidoConfirm = null;
    for (int i = 0; i < CantPedidos; i++) {
        if (listaPedidos [i].getIdPedido ().equalsIgnoreCase (idPedido)) {
            indPedido = i;
            PedidoConfirm = listaPedidos [i];
        }
    }
    if (PedidoConfirm == null) {
        try {
            if (PedidoConfirm.getProd () == null) { // Verifico que los productos no sean nulos
                for (int i = 0; i < PedidoConfirm.getCartProd (); i++)
                    PedidoConfirm.getProd (i).getProd ();
                if (PedidoConfirm.getProd (0) == null) //verificamos el item y el producto
                    AdminIDProductos.ReservarStock (PedidoConfirm.getProd (0).getCodigo ());
                else {
                    System.out.println ("Error producto nulo en el indice " + idPedido + " en el indice " + i);
                }
            }
            AdminIDProductos.guardarProductos (); //guardemos cambios del stock
        } catch (Exception e) {
            Bitacora.RegistrarEvento (Bitacora.Tipo_Vendedor, vendedor.getId (), Bitacora.OP_Confirmar_Pedido, Bitacora.ESTADO_FALLIDA, "No se pudo confirmar la compra");
            System.out.println ("Error el procesar productos del pedido " + idPedido + " " + e.getMessage ());
            e.printStackTrace (); // Imprime más detalles del error
            JOptionPane.showMessageDialog (null, "Error el procesar los productos del pedido. Verifica la consola.", "Error Interno", JOptionPane.ERROR_MESSAGE);
            return false;
        }
    }
    //le subimos las ventas al vendedor
    vendedor.setVentasHechas (vendedor.getVentasHechas () + 1);
    vendedor.setTotalVentasHechas (vendedor.getVentasHechas () + PedidoConfirm.getTotal ());
    AdminIDVendedores.guardarVendedor ();
    //eliminamos el pedido de la lista de pendientes
    for (int i = indPedido; i < CantPedidos - 1; i++) {
        listaPedidos [i] = listaPedidos [i + 1];
    }
    listaPedidos [CantPedidos - 1] = null;
    guardarEstado ();
    JOptionPane.showMessageDialog (null, "El pedido " + idPedido + " ha sido confirmado \nLa cantidad disponible se ha actualizado ");
    return true;
}
return false;
```

En este método primeramente hacemos declaraciones para el índice del pedido para luego con un ciclo recorrer la cantidad de pedidos y la lista de los mismos para buscar el código del pedido, al ser encontrado verificamos que los productos no sean nulos volvemos a recorrer para verificar otra vez el producto hacemos llamado a la reserva de Stock para luego restar la cantidad pedida del Stock actual y guardarlo en la materia de productos, siguiendo se le aumenta las ventas al vendedor haciendo llamado al constructor de vendedores y haciendo llamado al total de ventas para luego guardar estos datos en la matriz de vendedores en el apartado de ventas, hacemos otro llamado al constructor de productos del carrito para confirmar el pedido para de esta manera anotar los datos de la misma, que es el código del pedido, la hora de generación, el código del cliente, el total de la compra, los compras que fueron confirmadas, y la cantidad de productos, haciendo último llamado para la administrador de

compras y agregar una nueva compra, como último eliminamos el pedido de la tabla de pedidos y por lo consiguiente de la lista de los mismos.

Administrador de compras

```
public class AdminDCompras {  
  
    private static final String ArchivCompra = "HistorialCompras.csv"; // archivo de las compras  
    private static final int MaxCompras = 500; //el maximo de compras confirmadas gracias a lanzar moneda online  
    private static CompraAceptada[] listaCompras = new CompraAceptada[MaxCompras];  
    private static int CntCompras = 0;  
    private static final String ESTADO_COMPRAS_SER = "ComprasAceptadas.ser";
```

En esta clase contiene todos lo relacionado con las compras realizadas por los clientes, inicialmente tenemos lo que es de historial de compras en el archivo csv, y luego tenemos la siguiente constante que es el otro archivo de “comprasaceptadas.ser,” y tenemos la habitual la cantidad la matriz y la lista de compras

Carga de compras

```
public static void CargarCompra () {  
    File archivo = new File(ArchivCompra);  
    if (archivo.exists()) {  
        return; //si no hay historial  
    }  
    CntCompras = 0;//reiniciar el contador antes de cargar  
    try (BufferedReader lector = new BufferedReader(new FileReader(archivo))) {  
        String linea;  
        lector.readLine();  
  
        while ((linea = lector.readLine()) != null && CntCompras < MaxCompras) {  
            String[] datos = linea.split(",");  
            if (datos.length >= 4) {  
                try {  
                    String idPedido = datos[0].trim();  
                    LocalDateTime fecha = LocalDateTime.parse(datos[1].trim());  
                    String idCliente = datos[2].trim();  
                    double total = Double.parseDouble(datos[3].trim());  
  
                    Pedidos pedido = new Pedidos(idPedido, fecha, idCliente, "Cliente Desconocido", total, null, 0);  
                    listaCompras [CntCompras ++] = new CompraAceptada(idPedido, fecha, idCliente, total, null, 0);  
                } catch (Exception e) {  
                    System.out.println("Error al cargar linea de Compras.csv: " + linea + " -> " + e.getMessage());  
                }  
            } else {  
            }  
        }  
    } catch (IOException e) {  
        JOptionPane.showMessageDialog(null, "Error al cargar el historial de compras.", "Error", JOptionPane.ERROR_MESSAGE);  
    }  
}  
//para guardar en el csv
```

En este método hacemos llamado primeramente el archivo correspondiente al historial de compras, para luego leer el archivo y luego insertar los datos correspondientes a: código del pedido, la fecha de generación, el código del cliente y el total de la compra, para luego hacer llamado de la lista y la matriz de compras para añadir esta nueva la compra a la matriz.

Guardar Compra

```
public static void GuardarCompras () {
    File archivo = new File(ArchivCompra);
    try (PrintWriter escribir = new PrintWriter(new FileWriter(archivo))) {
        //para el encabeza
        escribir.println("idPedido,fechaConfirmacion,idCliente,total");
        for (int i = 0; i < CantCompras; i++) {
            CompraAceptada c = listaCompras [i];
            //guardar la fecha en formato para facilitar la lectura
            String lineaCsv = c.getIdPedido() + "," + c.getFechaConfirm().toString() + "," + c.getIdCliente() + "," + c.getTotal();
            escribir.println(lineaCsv);
        }
    } catch (IOException e) {
        System.err.println(" ERROR FATAL al guardar ComprasConfirmadas.csv:");
        e.printStackTrace();
        JOptionPane.showMessageDialog(null,
            "No se pudo guardar el historial de compras.\nError: " + e.getMessage() + "\nVerifica los permisos de la carpeta.",
            "Error de Archivo",
            JOptionPane.ERROR_MESSAGE);
    }
    guardarEstado ();
    GuardarCompras ();
}
```

En este método hace hemos llamado a el archivo de historial de compras para luego leer y escribir sobre él de esta manera escribiéndole encabezado que corresponde a: código del pedido, fecha de confirmación, código del cliente, total de compra, para luego recorrer la matriz de compras para añadirla los datos que se mencionaron anteriormente para luego guardarla en la matriz.

Agregar compra

```
//metodo para agregar una compra al arreglo
public static void agregarCompra (CompraAceptada compra) {
    if (CantCompras < MaxCompras) {
        listaCompras [CantCompras ++]= compra;
        //en el futuro PODRIA poner algo como pa guardar en un csv, pero allí veo xd
        //efectivamente lo hice xd
        GuardarCompras ();//lo guardamos de una
    } else {
        JOptionPane.showMessageDialog(null, "Ha alcanzado el maximo de compras", "Aviso", JOptionPane.WARNING_MESSAGE);
    }
    guardarEstado ();
}
```

Este método se encarga de de la lista de compras y la materia de las mismas guardar cada compra realizada en la materia correspondiente.

Datos de la tabla del historial de compras

```
public static Object[][] DatosTablaHistorial (String idCliente) {  
  
    //contamos cuantas compras tiene el cliente  
    int contadorCliente = 0;  
    for (int i = 0; i < CantCompras ; i++) {  
        if (listadCompras [i] != null && listadCompras [i].getIdCliente().equalsIgnoreCase(idCliente)) {  
            contadorCliente++;  
        }  
    }  
    Object[][] datos = new Object[contadorCliente][3]; //codifo, fecha y total ein quesalitos  
    DateTimeFormatter formatoVista = DateTimeFormatter.ofPattern ("yyyy-MM-dd HH:mm:ss");  
    int indiceDat = 0;  
  
    //recorremos la lista completa y copiamos solo las del cliente  
    for (int i = 0; i < CantCompras ; i++) {  
        CompraAceptada c = listadCompras [i];  
        if (c.getIdCliente().equalsIgnoreCase(idCliente)) {  
            datos[indiceDat][0] = c.getIdPedido();  
            datos[indiceDat][1] = c.getFechaConfirm().format(formatoVista);  
            datos[indiceDat][2] = c.getTotal();  
            indiceDat++;  
        }  
    }  
    return datos;  
}
```

En este método hacemos un contador para el cliente para luego recorrer la cantidad de compras y la lista de las mismas para obtener el código del cliente de esta manera aumentamos la cantidad de compras que ha hecho al cliente, siguiente hacemos relación de un objeto del contador de compras del cliente y él tiene sólo tres espacios que corresponde al código, fecha y el total de la compra, siguiendo recorremos otra vez la cantidad de compras para concretar con las compras ya realizadas de esta manera obtenemos los datos del código del pedido, la fecha de confirmación y el total de la compra.

Buscar actualizaciones de Stock

```
1 public static String BuscarActualizacionStock (String codigoProducto) {
2     File ArchivHist = new File("HistorialStock.csv");
3     LocalDateTime ultFecha = null; //última fecha de actualización
4     DateTimeFormatter FormatoArchiv = DateTimeFormatter.ISO_LOCAL_DATE_TIME; //pa la fecha
5
6     if (!ArchivHist.exists()){
7         return "No hay historial"; //no necesito explicar
8     }
9     try (BufferedReader lector = new BufferedReader(new FileReader(ArchivHist))) {
10        String linea;
11        lector.readLine();
12
13        while ((linea = lector.readLine()) != null) {
14            String[] datos = linea.split(",");
15            //formato que esperamos Fecha en iso , código producto, cantidad, usuario.id-usuario,
16            if (datos.length >= 2 && datos[1].trim().equalsIgnoreCase(codigoProducto)) {
17                try {
18                    LocalDateTime fActual = LocalDateTime.parse(datos[0].trim(), FormatoArchiv);
19                    if (ultFecha == null || fActual.isAfter(ultFecha)) {
20                        ultFecha = fActual;
21                    }
22                } catch (Exception e) {
23                    System.err.println("Error al leer la fecha en historial: " + linea);
24                }
25            }
26        }
27    } catch (IOException e) {
28        System.err.println("Error al leer HistorialStock.csv " + e.getMessage());
29        return "Error de lectura";
30    }
31    if (ultFecha != null) {
32        //devolvemos la feyc en un formato mas leible para menos como yo xd
33        return ultFecha.format(DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm"));
34    } else {
35        return "sin registros"; // no se encontraron ingresos para el producto
36    }
37 }
```

Éste método se declara el archivo de historial de Stock para luego escribir en él lo que corresponde a puntos la fecha, código del producto, cantidad, usuario, código del usuario de esta manera obtener la última fecha de actualización de Stock para el reporte de Stock.

Administrador de clientes

```
public class AdminDClientes {

    private static final String ArchiClient = "Clientes.csv";
    private static final String ESTADO_CLIENTES_SER = "clientes.ser";
    private static final int MaxClientes = 100; //el maximo de clientes

    private static Cliente[] listadClientes = new Cliente[MaxClientes];
    private static int CantClientes = 0;
```

En esta clase contenemos todo lo relacionado a los clientes, inicialmente debemos declarado las constantes que corresponden al archivo de clientes tanto para el archivo CSV como para el archivo SER, también declaramos las constantes para la matriz la cantidad y el máximo de clientes.

Cargar clientes

```
public static void CargarClientes () {
    File archivo = new File(ArchiClient);
    if (!archivo.exists()) {
        return; // si no hay un archivo no deberia cargarse nada
    }
    CantClientes = 0;
    try (BufferedReader lector = new BufferedReader(new FileReader(archivo))) {
        String linea;
        int num linea = 1;
        lector.readLine(); // nos saltamos el encabezado

        while ((linea = lector.readLine()) != null && CantClientes < MaxClientes) {
            String[] datos = linea.split(",");
            if (datos.length == 5) { // id, nombre, contraseña, genero, cumple
                Cliente cliente = new Cliente(
                    datos[0].trim(),
                    datos[1].trim(),
                    datos[2].trim(),
                    datos[3].trim(),
                    datos[4].trim()
                );
                listadClientes [CantClientes ++] = cliente;
            }
        }
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Error al cargar el archivo de clientes.", "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

En este método se hemos llamado al archivo correspondiente de clientes para luego hacer una validación para la cantidad de clientes para luego hacer una lectura del archivo de clientes de esta manera podemos hacer un bucle que voy a hacer hasta la cantidad máxima de clientes llenando el archivo CSV con los datos correspondientes a: código, nombre, contraseña, género, cumpleaños aumentando la lista de clientes y la cantidad de estos.

Guardar clientes Y código repetido

```
public static void GuardarClientes () {
    System.out.println("Guardando Clientes.csv"); // Mensaje debug
    try (PrintWriter escribir = new PrintWriter(new FileWriter(ArchiClient ))) {
        escribir.println("Codigo,nombre,contraseña,genero,cumpleaños");//el encabezado
        //ciclo for para recorrer la lista de clientes creados para encontrar los que se hayan creado ya
        for (int i = 0; i < CantClientes ; i++){
            Cliente c = listadClientes [i];
            if (c != null){
                String lineaCsv
                    = c.getId() + ","
                    + c.getNombre() + ","
                    + c.getContraseña() + ","
                    + c.getGenero() + ","
                    + c.getCumpleaños();
                escribir.println(lineaCsv);
            }
        }
        System.out.println("Clientes.csv guardado"); // Mensaje debug
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, "Error al guardar los clientes en el archivo.", "Error", JOptionPane.ERROR_MESSAGE);
    }
    cargarEstado ();
}

//veremos si el codigo del cliente ya essta en uso
public static boolean CodRepetido (String id) {
    for (int i = 0; i < CantClientes ; i++){
        if (listadClientes [i].getId().equalsIgnoreCase(id)) {//ignoramos las mayusculas y minusculas
            return true;
        }
    }
    return false;
}
```

Para el primer método hacemos llamado otra vez el archivo de clientes para escribir sobre el usando print writer, escribimos el encabezado que corresponde a código, nombre, contraseña, género, cumpleaños, para luego recorrer un ciclo que vaya desde cero hasta la cantidad de clientes llenando el archivo se sube con todos los clientes creados, el código repetido se encarga de buscar en la materia de clientes en el listado de clientes por alguna igualdad en el código de algún cliente.

Creación de clientes

```
//lo mismo de productos y vendedores hasta el nombre lo copie, es
public static boolean CreacionClientes (String id, String nombre, String contraseña, String genero, String cumpleaños, String idVendedor) {
    if (CantClientes >= MaxClientes) {
        JOptionPane.showMessageDialog (null, "Se alcanzó el límite de clientes para crear", "Error", JOptionPane.ERROR_MESSAGE);
        Bitacora.RegistrarEvento (Bitacora.Tipo_Vendedor , idVendedor, Bitacora.OP_Crear_Cliente , Bitacora.ESTADO_FALLIDA , "se llego al limite de clientes creados");
        return false;
    }
    if (CodRepetido (id)) {
        JOptionPane.showMessageDialog (null, "El código de cliente ya está en uso", "Error", JOptionPane.ERROR_MESSAGE);
        Bitacora.RegistrarEvento (Bitacora.Tipo_Vendedor , idVendedor, Bitacora.OP_Crear_Cliente , Bitacora.ESTADO_FALLIDA , "no se creo el cliente por codigo repetido");
        return false;
    }
    //Creamos el nuevo cliente
    Cliente Ncliente = new Cliente(id, nombre, contraseña, genero, cumpleaños);
    listadClientes [CantClientes ++] = Ncliente;

    //Lo agregamos ambien a la lista pa que LUEGO INICIEN SESION NO PIENSO K GARLA DE NUEVO
    AdminUsuarios.AgregarUsuario (Ncliente);

    //y lo guardamos en la lisata de clientes xd
    GuardarClientes ();
    Bitacora.RegistrarEvento (Bitacora.Tipo_Vendedor , idVendedor, Bitacora.OP_Crear_Cliente , Bitacora.ESTADO_EXITOSA , "se ha creado un cliente");
    return true;
}
```

En este método hacemos llamado el constructor de clientes para luego hacer una validación si se alcanzó el límite de clientes creados para luego hace una validación sobre si el código de cliente ya está repetido, creamos el cliente con unos parámetros de código, nombre, contraseña, género, cumpleaños para luego añadirlo a la lista y aumentando el contador de clientes, también lo agregamos al estado de usuarios para luego guardar los clientes de la matriz de clientes.

Tabla de datos de cliente y buscar cliente

```
//para la tabla de Clientes LPM
public static Object[][] DatosTablaCliente () {
    //miramos si hay vendedores cargados para evitar errores
    if (listadClientes == null || CantClientes == 0){
        return new Object[0][4];
    }
    Object[][] datos = new Object[CantClientes][4];
    for (int i = 0; i < CantClientes ; ++){
        //obtenemos el objeto vendedor de la lista
        Cliente c = listadClientes [i];
        datos[i][0] = c.getId();
        datos[i][1] = c.getNombre();
        datos[i][2] = c.getGenero();
        //ESTO IMPEDIA QUE LA MALDITA TABLA SE ACTUALICE
        datos[i][3] = c.getCumpleaños();
    }
    return datos;
}

//metodo pa buscar al cliente por su codigo
public static Cliente BuscarCliente (String id) {
    for (int i = 0; i < CantClientes ; ++){
        if (listadClientes [i].getId().equalsIgnoreCase(id)){
            return listadClientes [i]; //si lo encontramos devuelve el objeto
        }
    }
    return null; //devuelve nulo o mejor díjico no devuelve na' si no lo encuentra
}
```

El primer método se encarga de crear un objeto que tenga la cantidad de cuatro que corresponde a: el código del cliente, el nombre del cliente, el género, el cumpleaños de este, el otro método se encarga de buscar el cliente en la matriz de clientes buscando por el código devolviendo el cliente con el código buscado.

Modificar cliente

```
//pa modificar clientes
public static boolean ModCliente (String id, String Nnombre, String Ncontraseña, String Ngenero, String Ncumple, String idVendedor) {
    //buscamos al cliente que queremos modificar
    Cliente Modcliente = BuscarCliente (id);

    if (Modcliente != null) {
        //si lo encontramos, actualizamos sus datos con los setters o serers como hablan en inglich
        Modcliente.setNombre(Nnombre);
        Modcliente.setContraseña(Ncontraseña);
        Modcliente.setCumpleaños(Ncumple);
        Modcliente.setGenero(Ngenero);
        //esto es para que el cambio de contraseña funcione en el inicio de sesion
        AdminDUsuarios.ActualizarUsuario (id, Nnombre, Ncontraseña);

        //lo guardamos en el archivo csv de clientes
        GuardarClientes ();
        Bitacora.RegistrarEvento (Bitacora.Tipo_Vendedor , idVendedor, Bitacora.OP_Mod_Cliente , Bitacora.ESTADO_EXITOSA , "Se modifoco los datos correctamente de un cliente");

        return true;//devolvemos un verdadero si salio bien xd
    }
    Bitacora.RegistrarEvento (Bitacora.Tipo_Vendedor , idVendedor, Bitacora.OP_Mod_Cliente , Bitacora.ESTADO_FALLIDA , "No se encontro al cliente");

    return false;
}
```

En este método queremos un cliente y lo buscamos por su código para luego cargar los atributos que corresponden a: el nombre, contraseña, cumpleaños, género para esta manera poder agregar nuevos datos luego se actualiza el usuario haciendo el llamado a el administrador de usuarios, lo guardamos en la materia de clientes también.

Eliminar cliente

```
public static boolean EliminarCliente (String Id, String idVendedor) {
    int encontrar = -1;
    //buscamos el indice del cliente a eliminar
    for (int i = 0; i < CantClientes ; i++){
        if (listadClientes [i].getId().equalsIgnoreCase(Id)) {
            encontrar = i;
            break;
        }
    }
    if (encontrar != -1) {
        //eliminamos al cliente de la lista de clientes
        for (int i = encontrar; i < CantClientes - 1; i++){
            listadClientes [i] = listadClientes [i + 1];
        }
        listadClientes [CantClientes - 1] = null;
        CantClientes --;

        //lo eliminamos de la lista general de usuarios
        AdminDUsuarios.EliminarUsuario (Id);

        //lo guardamos en ambos archivos para mantener la consistencia
        GuardarClientes ();
        AdminDUsuarios.GuardarUsuarios (); //ca que se guarde en la parte de inicio de sesion
        Bitacora.RegistrarEvento (Bitacora.Tipo_Vendedor , idVendedor, Bitacora.OP_Eli_Cliente , Bitacora.ESTADO_EXITOSA , "Se elimino correctamente a un cliente");

        return true;
    }
    return false; //si no encuentra al cliente xd
}
```

En este método recorre la matriz de clientes buscando el código del cliente que queremos buscar, para luego al encontrarlo eliminar a este cliente de la matriz eliminamos tanto de matriz de clientes como de la matriz de usuarios.

Cargar clientes

```
public static String CargarClientes(File archivo, String idVendedor) {
    StringBuilder resumen = new StringBuilder();
    int cargoExito = 0;
    int fallaForm = 0;
    int fallaLogic = 0; //para los duplicados y otras chingaderas
    boolean cambios = false;

    try (BufferedReader lector = new BufferedReader(new FileReader(archivo))) {
        lector.readLine();
        int numLinea = 1;

        String linea;
        while ((linea = lector.readLine()) != null) {
            numLinea++;

            if (linea.trim().isEmpty()) {
                continue;
            }
            String[] datos = linea.split(",");
            //validacion que sean 5 columnas
            if (datos.length != 5) {
                resumen.append("linea").append(numLinea).append(": error porque se esperan 5 columnas.\n");
                fallaLogic++;
                continue;
            }

            String codigo = datos[0].trim();
            String nombre = datos[1].trim();
            String contraseña = datos[2].trim().toUpperCase();
            String genero = datos[3].trim();
            String cumple = datos[4].trim();

            // otra validacion si el codigo ya existe
            if (!AdminClientes.CodRepetido(codigo)) {
                resumen.append("linea").append(numLinea).append(": error porque el codigo : " + codigo.append(" esta repetido \n"));
                Bitacora.RegistrarEvento(Bitacora.Tipo_Vendedor, idVendedor, Bitacora.OP_Cargar_Cliente, Bitacora.ESTADO_FALLIDA, "No se cargaron correctamente los clientes");
            } else {
                cambios = true;
                cargoExito++;
            }
            resumen.append("linea").append(numLinea).append(": error al crear al cliente.\n");
            fallaLogic++;
        }
    } catch (IOException e) {
        return "Error al leer el archivo " + e.getMessage();
    }
    if (cambios) {
        GuardarClientes();
        guardarEstado();
    }
    String respFinal = "Resumen de la carga de clientes\n";
    respFinal += "Clientes cargados correctamente: " + cargoExito + "\n";
    respFinal += "Errores de formato: " + fallaForm + "\n";
    respFinal += "Duplicados y otros: " + fallaLogic + "\n";
    respFinal += "Detalles:\n" + resumen.toString();
    Bitacora.RegistrarEvento(Bitacora.Tipo_Vendedor, idVendedor, Bitacora.OP_Cargar_Cliente, Bitacora.ESTADO_EXITOSA, "Se cargaron correctamente los clientes");
    return respFinal;
}
```

En este método hacemos llamado a el archivo de clientes, de carambola variables para carga exitosa falla de formato falla lógica y por si hubo cambios, leemos el archivo y empecemos a llenar el archivo de clientes con los datos de: código, nombre, contraseña, género, cumpleaños añadimos validaciones y el código ya está repetido, y creamos un cliente sin guardar para luego guardar esto en la materia de clientes y mostrar un resumen de la carga de los clientes.

Creación de clientes sin guardar

```
public static boolean CreacionClientesSinGuardar(String id, String nombre, String contraseña, String genero, String cumpleaños, String idVendedor) {
    if (CantClientes >= MaxClientes) {
        return false;
    }
    Cliente Ncliente = new Cliente(id, nombre, contraseña, genero, cumpleaños);
    listaClientes [CantClientes ++] = Ncliente;

    if (!AdminDUsuarios.CodRepetido (id)) {
        AdminDUsuarios.AgregarUsuario (Ncliente);
    } else {
        AdminDUsuarios.ActualizarUsuario (id, nombre, contraseña);
        return true;
    }
    return false;
}
```

Este método nos permite crear un cliente sin tener que guardarla en una matriz de clientes de igual manera de una validación del máximo de clientes y quedamos el cliente con: su código, su nombre, su contraseña, su género, su cumpleaños aumentando sólo la cantidad de clientes y agregándolo a la matriz y el archivo CSV de usuarios.

BITACORA

```

public class Bitacora {

    private static final String Archiv_Bitacora = "Bitacora.csv";
    private static final DateTimeFormatter FHFomato = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");

    //Constantes para los usuarios
    public static final String Tipo_Admin = "ADMIN";
    public static final String Tipo_Vendedor = "VENDEDOR";
    public static final String Tipo_Cliente = "CLIENTE";
    public static final String Tipo_Sistema = "SISTEMA";//para los eventos que no sea involucrado a los webones de los usuarios

    //Constantes para las operaciones
    public static final String OP_Ingreso_Exito = "INGRESO_EXITOSO";
    public static final String OP_Ingreso_Fallido = "INGRESO_FALLIDO";
    public static final String OP_Salida = "SALIDA";

    //para lo del vendedor
    public static final String OP_Crear_Vendedor = "CREAR_VENDEDOR";
    public static final String OP_Mod_Vendedor = "MODIFICAR_VENDEDOR";
    public static final String OP_Eli_Vendedor = "ELIMINAR_VENDEDOR";
    public static final String OP_Cargar_Vendedor = "CARGAR_VENDEDOR_CSV";

    //para lo del precurso
    public static final String OP_Crear_Producto = "CREAR_PRODUCTO";
    public static final String OP_Mod_Producto = "MODIFICAR_PRODUCTO";
    public static final String OP_Eli_Producto = "ELIMINAR_PRODUCTO";
    public static final String OP_Cargar_Producto = "CARGAR_PRODUCTO";
    public static final String OP_Agregar_Stock = "AGREGAR_STOCK";
    public static final String OP_Ver_Detalles_Prod = "VER_DETALLES_PRODUCTO";
    public static final String OP_Cargar_Stock = "CARGAR_STOCK_CSV";

    //PARA EL CLIENTE
    public static final String OP_Crear_Cliente = "CREAR_CLIENTE";
    public static final String OP_Mod_Cliente = "MODIFICAR_CLIENTE";
    public static final String OP_Eli_Cliente = "ELIMINAR_CLIENTE";
    public static final String OP_Cargar_Cliente = "CARGAR_CLIENTE_CSV";

    //otroescosas
    public static final String OP_Agregar_Producto_Carrito = "AGREGAR_PRODUCTO_CARRITO";
    public static final String OP_Actualizar_Pedido = "ACTUALIZAR_PEDIDO";
    public static final String OP_Eliminar_Producto_Carrito = "ELIMINAR_PRODUCTO_CARRITO";
    public static final String OP_Realizar_Pedido = "REALIZAR_PEDIDO";
    public static final String OP_Confirmar_Pedido = "CONFIRMAR_PEDIDO";
    public static final String OP_Ver_Historial = "VER_HISTORIAL_COMPRA";

    //reportes
    public static final String OP_Generar_Reporte = "GENERAR_Reporte";
    public static final String OP_Generar_Reporte_Prod_Mas = "GENERAR_Reporte_PRODUCTOS_MAS_VENDIDOS";
    public static final String OP_Generar_Reporte_Prod_Menos = "GENERAR_Reporte_PRODUCTOS_MENOS_VENDIDOS";
    public static final String OP_Generar_Reporte_Inventario = "GENERAR_Reporte_INVENTARIO";
    public static final String OP_Generar_Reporte_Ventas_Vendedor = "GENERAR_Reporte_VENTAS_VENDEDOR";
    public static final String OP_Generar_Reporte_Clientes_Activos = "GENERAR_Reporte_CLIENTES_ACTIVOS";
    public static final String OP_Generar_Reporte_Financiera = "GENERAR_Reporte_FINANCIERO";
    public static final String OP_Generar_Reporte_Prod_Caducar = "GENERAR_Reporte_PRODUCTOS POR CADUCAR";

    public static final String OP_GENERAR_Reporte = "GENERAR_Reporte";
    public static final String OP_EXPORTAR_BITACORA = "EXPORTAR_BITACORA";

    // estados de la tarea
    public static final String ESTADO_EXITOSA = "EXITOSA";
    public static final String ESTADO_FALLIDA = "FALLIDA";
}

```

En esta clase y método está relacionado con todo lo que tenga que ver con la bitácora creamos constantes que hagan solución a el estado de la tarea a los diferentes tipos de usuarios y todas las tareas que se pueden realizar.

Registrar Accion

```

public static synchronized void RegistrarEvento(String tipoUsuario, String codigoUsuario, String operacion, String estado, String descripcion) {
    LocalDateTime ahora = LocalDateTime.now();
    String FHFomato = ahora.toString();
    //limpiamos y preparamos los datos para el csv
    String tipoUsulImpio = (tipoUsuario != null ? tipoUsuario.toUpperCase() : "DESCONOCIDO").replace("\n", "\\\n");
    String codigoUsulImpio = (codigoUsuario != null ? codigoUsuario.toUpperCase() : "\\\\n").replace("\n", "\\n");
    String opImpio = (operacion != null ? operacion.toUpperCase() : "DESCONOCIDA").replace("\n", "\\n");
    String estadoImpio = (estado != null ? estado.toUpperCase() : "DESCONOCIDO").replace("\n", "\\n");
    String descImpio = (descripcion != null ? descripcion : "").replace("\n", "\\n");
    //para escribir en el archivo

    String lineaCsv = String.format("%s,%s,%s,%s,%s,%s", FHFomato,
        tipoUsulImpio,
        codigoUsulImpio,
        opImpio,
        estadoImpio,
        descImpio);
    File archivo = new File(Archiv_Bitacora);
    boolean esNuevo = !archivo.exists() || archivo.length() == 0;
    try (PrintWriter escribir = new PrintWriter(new FileWriter(Archiv_Bitacora, true))) {
        if (esNuevo) {
            escribir.println("FechaHoraISO,TipoUsuario,CodigoUsuario,Operacion,Estado,Descripcion");
        }
        escribir.println(lineaCsv);
    } catch (IOException e) {
        System.err.println("No se pudo escribir en el archivo de bitácora");
        System.err.println("Línea no registrada: " + lineaCsv);
        e.printStackTrace();
    }
}

```

Éste método se encarga de registrar las acciones que el usuario haga con el formato de, fecha, usuario, código de usuario, operación, estado de la tarea, descripción de la tarea, esto lo va a escribir en un archivo CSV y escribe el encabezado con los datos mencionados anteriormente.

Bitacora en pdf

```
//para exportar esta chingada a pdf
public static void BitacoraPdf() {
    File ArchivoCsv = new File("Archiv_Bitacora");
    if (!ArchivoCsv.exists() || ArchivoCsv.length() == 0) {
        JOptionPane.showMessageDialog(null, "El archivo de la bitacora esta vacio o no existe", "Aviso", JOptionPane.INFORMATION_MESSAGE);
        return;
    }
    LocalDateTime hora = LocalDateTime.now();
    DateTimeFormatter Formatonombre = DateTimeFormatter.ofPattern("dd_MM_yyyy_HH_mm_ss");
    String NombreArchiv = hora.format(Formatonombre) + "_Bitacora.pdf";

    try {
        PdfWriter escribir = new PdfWriter(NombreArchiv);
        PdfDocument pdf = new PdfDocument(escribir);
        Document document = new Document(pdf);

        document.add(new Paragraph("Bitácora de acciones"))
            .setFontSize(16).setBold().setHorizontalAlignment(TextAlignment.CENTER);
        document.add(new Paragraph("Exportado el: " + hora.format(FHFormato))
            .setHorizontalAlignment(TextAlignment.CENTER).setMarginBottom(15));
        Table tabla = new Table(UnitValue.createPercentArray(new float[]{18, 12, 12, 18, 10, 30})); // Ajusta anchos
        tabla.setWidths(new float[]{18, 12, 12, 18, 10, 30}); // useAllAvailableWidth();
        //la tabla con las 6 columnas

        for (Cell celda : celdas) {
            if (encabezado) {
                tabla.addCellHeaderCell(celda);
            } else {
                tabla.addCell(celda);
            }
        }
        else_if (linea.trim().isEmpty()) { //no añadimos filas vacias
            tabla.addCell(new Cell(1, 6).add(new Paragraph("Error al parsear la linea: " + linea).setFontSize(8)));
        }
        encabezado = false;
    }
    document.add(tabla);
    document.close();
    RegistrarEvento(Bitacora.Tipo_Admin, "admin", Bitacora.OP_EXPORTAR_BITACORA, Bitacora.ESTADO_EXITOSA, "Bitacora exportada a PDF");
    JOptionPane.showMessageDialog(null, "Bitacora exportada a pdf", "Exportación Completa", JOptionPane.INFORMATION_MESSAGE);
} catch (IOException e) {
    System.err.println("Error al exportar la bitácora a PDF: " + e.getMessage());
    e.printStackTrace();
    RegistrarEvento(Bitacora.Tipo_Admin, "admin", Bitacora.OP_EXPORTAR_BITACORA, Bitacora.ESTADO_FALLIDA, "No se pudo exportar la bitacora a PDF");
    JOptionPane.showMessageDialog(null, "Error al exportar la bitácora a PDF.", "Error", JOptionPane.ERROR_MESSAGE);
}
}

//leemos el csv y lo añadimos a pdf
try (BufferedReader lector = new BufferedReader(new FileReader(ArchivoCsv))) {
    String linea;
    boolean encabezado = true;
    while ((linea = lector.readLine()) != null) {
        String[] partes = linea.split(",\\=\\\"|\\\"\\\"|\\\"\\\"\\\"|\\\"\\\"\\\"\\\"\\$\\\", -1); // Split por coma fuera de comillas, inteligent no?
        if (partes.length == 6) {
            Cell[] celdas = new Cell[6];
            for (int i = 0; i < 6; i++) {
                // quitamos las comillas si existen al principio o al final
                String textCelda = partes[i];
                if (textCelda.startsWith("\"") && textCelda.endsWith("\"")) {
                    textCelda = textCelda.substring(1, textCelda.length() - 1).replace("\\"", "\"");//quitamos las comillas
                }
            }
            if (i == 0 && encabezado) {
                try {
                    LocalDateTime fecha = LocalDateTime.parse(textCelda);
                    textCelda = fecha.format(FHFormato);
                } catch (Exception e) {
                    //no gacemos nada
                }
            }
            celdas[i] = new Cell().add(new Paragraph(textCelda));
            if (encabezado) {
                celdas[i].setBold(); //todo en negrita
            }
        }
    }
}
```

Éste método se encarga de ser llamado al archivo de la bitácora declaramos el nombre del archivo que es el la fecha y hora actual junto al nombre te acuerdas. PDF, declaramos el documento PDF y le agregamos lo que es el título el subtítulo y agregamos una tabla donde irán todos los datos recopilados de la bitácora, hacemos llamado otra vez a los al archivo CSV y lo llenamos con los datos mencionados anteriormente cerramos el documento y añadimos la tabla.

Borrar Bitacora

```
public static void BorrarBitacora () {
    File archivoBit = new File(Archiv_Bitacora);

    if (archivoBit.exists()){
        JOptionPane.showMessageDialog(null, "El archivo de la bitacora no existe actualmente.", "Archivo No Encontrado", JOptionPane.INFORMATION_MESSAGE);
        return;
    }

    //pedimos la confirmacion
    int Confirm = JOptionPane.showConfirmDialog(null, "Estas seguro de querer borrar el archivo de la bitacora?", "Confirmar borrado", JOptionPane.YES_NO_OPTION, JOptionPane.WARNING_MESSAGE
    //si el usuario dice si
    if (Confirm == JOptionPane.YES_OPTION){
        //intentemos borrar
        try {
            RegistrarEvento(Bitacora.Tipo_Admin, "admin", "BORRAR_BITACORA", "INTENTO", "Intentando borrar la bitacora");

            if (archivoBit.delete()){
                JOptionPane.showMessageDialog(null, "Se elimino la bitácora", "Borrado completo", JOptionPane.INFORMATION_MESSAGE);

            } else {
                JOptionPane.showMessageDialog(null, "No se pudo borrar el archivo de bitácora", "Error al Borrar", JOptionPane.ERROR_MESSAGE);
            }
        } catch (SecurityException e){
            JOptionPane.showMessageDialog(null, "No se pudo borrar el archivo de bitácora", "Error al Borrar", JOptionPane.ERROR_MESSAGE);
        }
        } catch (Exception e){
            JOptionPane.showMessageDialog(null, "Ocurrió un error inesperado al borrar la bitácora.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    } else {
        JOptionPane.showMessageDialog(null, "Borrado de bitácora cancelada", "Cancelado", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Este método se encarga de eliminar los registros de la bitácora inicialmente hemos llamado otra vez al archivo de la bitácora y con la opción de un mensajero desplegable podemos eliminar o no la bitácora de acciones con diferentes mensajes dependiendo lo que pase si se eliminó si no se pudo si se canceló o si hubo algún error en la eliminación de la bitácora.