Universidad de San Carlos de Guatemala
Escuela de Ciencias y Sistemas
Facultad de Ingeniería
Introducción a la programación y Computación 1
Segundo Semestre 2025
Catedrático: William Escobar

MANUAL TÉCNICO

Gabriel Eduardo Urbina Sunún
Carnet: 202300384
Cui: 2794761080101

INTRODUCCIÓN

Su propósito principal es desarrollar una aplicación de escritorio en Java, con interfaz gráfica, que funcione como un sistema de gestión integral para una tienda en línea, aplicando de manera práctica los fundamentos de la Programación Orientada a Objetos (POO) y el patrón arquitectónico Modelo–Vista–Controlador (MVC).

La aplicación tiene como objetivo permitir la administración completa de usuarios, productos, inventarios y pedidos, integrando además módulos específicos según el rol del usuario (Administrador, Vendedor o Cliente).

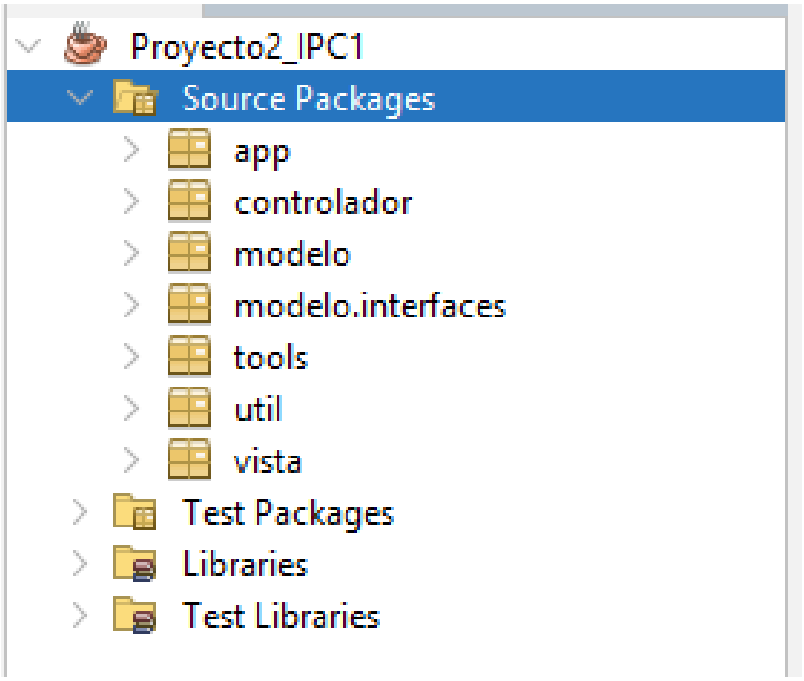
Entre sus principales funcionalidades se incluyen la autenticación segura, la gestión de vendedores, clientes y productos, la confirmación de pedidos, el control de inventario, la generación de reportes analíticos en formato PDF y el registro de eventos mediante una bitácora del sistema.

Asimismo, se incorpora el uso de hilos de ejecución (threads) para simular procesos automáticos en tiempo real, y la serialización de objetos para asegurar la persistencia de los datos entre sesiones.

REQUERIMIENTOS DEL SISTEMA

- Windows 11.
- Netbeans IDE 24.
- Intel core i5 de onceava generación.
- Tarjeta gráfica integrada.
- No tiene necesidad de una conexión a internet.
- Java
- Librería externa: iText 5.x para la generación de reportes en PDF.

EXPLICACIÓN

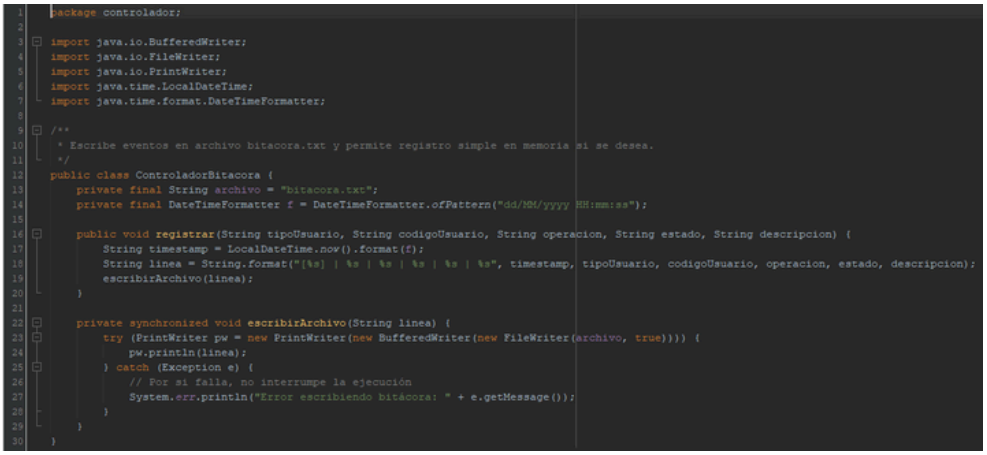


El programa contiene distintos package para un control más ordenado y mantener el modelo MVC (modelo, vista, controlador).
La primer parte del código que tenemos es el main, la línea 12 a 14 intercepta los posibles errores que se puedan dar al ejecutar el programa, en la línea 17 se inicializan los controladores, de la línea 20 a 43 se crean los monitores locales y se arrancan, verifica lo usuarios activos, los pedidos pendientes y las ventas del día. Después en el cogido solo se inicializan los hilos y se verifica que los monitores se detengan al terminar la jvm.

```
10  public class Main {
11      public static void main(String[] args) {
12          try {
13              UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
14          } catch (ClassNotFoundException | IllegalAccessException | InstantiationException | UnsupportedLookAndFeelException ignored) { }
15
16          // Inicializar controladores compartidos
17          final Controladores controladores = new Controladores();
18
19          // Crea monitores globales y los arranca inmediatamente
20          util.GlobalMonitorDispatcher dispatcher = util.GlobalMonitorDispatcher.getInstance();
21
22          util.HiloMonitor sesionesMonitor = new util.HiloMonitor("SesionesMonitor", 10_000, () -> {
23              int n = controladores.getUsuarioController().listarTodosUsuarios().length;
24              String ts = java.time.LocalDateTime.now().format(java.time.format.DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss"));
25              return String.format("Usuarios Activos: %d - Última actividad: %s", n, ts);
26          });
27          sesionesMonitor.setListener(dispatcher);
28
29          util.HiloMonitor pedidosMonitor = new util.HiloMonitor("PedidosPendientes", 8_000, () -> {
30              int k = controladores.getPedidoController().listarPedidosPendientes().length;
31              String ts = java.time.LocalDateTime.now().format(java.time.format.DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss"));
32              return String.format("Pedidos Pendientes: %d - Procesando... %s", k, ts);
33          });
34          pedidosMonitor.setListener(dispatcher);
35
36          util.HiloMonitor statsMonitor = new util.HiloMonitor("EstadisticasVivas", 15_000, () -> {
37              int ventas = 0;
38              for (modelo.Pedido p : controladores.getPedidoController().listarTodos()) if (p.isConfirmado()) ventas++;
39              int productos = controladores.getProductoController().listarProductos().length;
40              String ts = java.time.LocalDateTime.now().format(java.time.format.DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss"));
41              return String.format("Ventas del día: %d | Productos registrados: %d | %s", ventas, productos, ts);
42          });
43          statsMonitor.setListener(dispatcher);
44
45          // Arranca hilos
46          sesionesMonitor.start(); pedidosMonitor.start(); statsMonitor.start();
47
48          // Asegurar que los monitores se detengan al terminar la JVM
49          Runtime.getRuntime().addShutdownHook(new Thread(() -> {
50              try { sesionesMonitor.requestStop(); } catch (Exception ignored) {}
51              try { pedidosMonitor.requestStop(); } catch (Exception ignored) {}
52              try { statsMonitor.requestStop(); } catch (Exception ignored) {}
53          }));
```

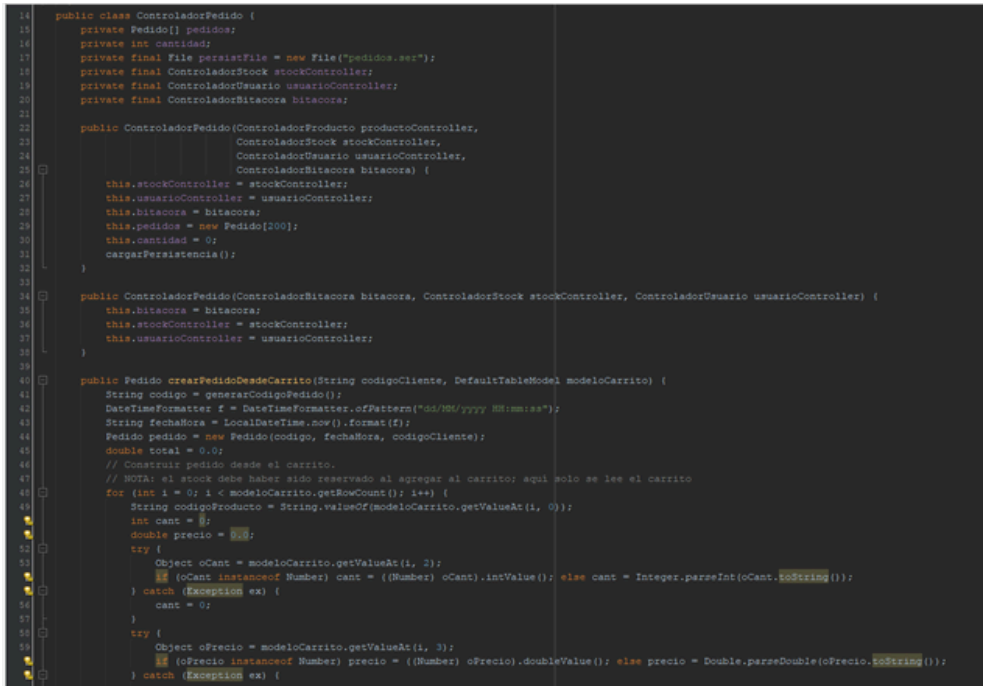
EXPLICACIÓN

Ahora en la sección de controladores tenemos la bitacora, en estos solo se escribe la bitácora en txt y se verifica si quiere un registro de memoria simple.



```java
package controlador;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

/**
 * Escribe eventos en archivo bitacora.txt y permite registro simple en memoria si se desea.
 */
public class ControladorBitacora {
    private final String archivo = "bitacora.txt";
    private final DateTimeFormatter f = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");

    public void registrar(String tipoUsuario, String codigoUsuario, String operacion, String estado, String descripcion) {
        String timestamp = LocalDateTime.now().format(f);
        String linea = String.format("[%s] | %s | %s | %s | %s | %s", timestamp, tipoUsuario, codigoUsuario, operacion, estado, descripcion);
        escribirArchivo(linea);
    }

    private synchronized void escribirArchivo(String linea) {
        try (PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(archivo, true)))) {
            pw.println(linea);
        } catch (Exception e) {
            // Por si falla, no interrumpe la ejecución
            System.err.println("Error escribiendo bitácora: " + e.getMessage());
        }
    }
}
```

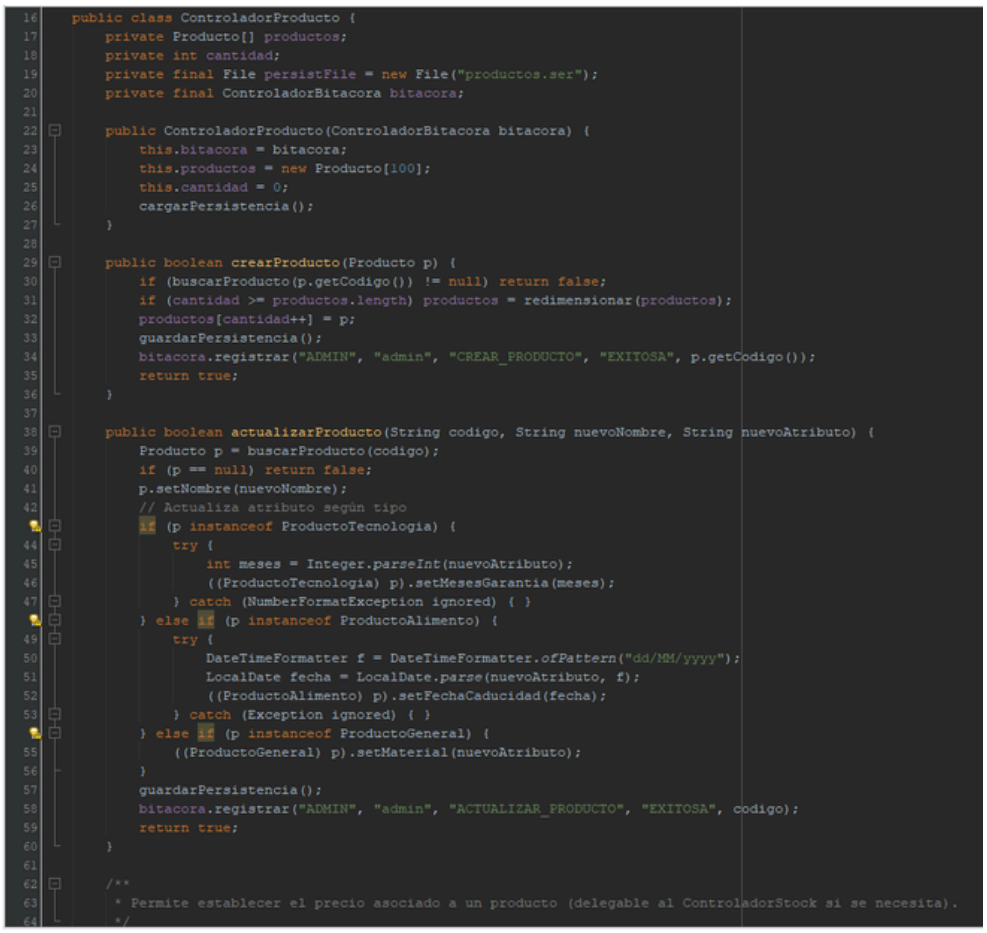
Ahora vamos con la sección de controlador de pedidos en las que se mantiene cola de pedidos pendientes en memoria y persistencia en pedidos.ser y también permite crear pedido desde un DefaultTableModel (carrito)



```java
public class ControladorPedido {
    private Pedido[] pedidos;
    private int cantidad;
    private final File persistFile = new File("pedidos.ser");
    private final ControladorStock stockController;
    private final ControladorUsuario usuarioController;
    private final ControladorBitacora bitacora;

    public ControladorPedido(ControladorProducto productoController,
                             ControladorStock stockController,
                             ControladorUsuario usuarioController,
                             ControladorBitacora bitacora) {
        this.stockController = stockController;
        this.usuarioController = usuarioController;
        this.bitacora = bitacora;
        this.pedidos = new Pedido[200];
        this.cantidad = 0;
        cargarPersistencia();
    }

    public ControladorPedido(ControladorBitacora bitacora, ControladorStock stockController, ControladorUsuario usuarioController) {
        this.bitacora = bitacora;
        this.stockController = stockController;
        this.usuarioController = usuarioController;
    }

    public Pedido crearPedidoDesdeCarrito(String codigoCliente, DefaultTableModel modeloCarrito) {
        String codigo = generarCodigoPedido();
        DateTimeFormatter f = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");
        String fechaHora = LocalDateTime.now().format(f);
        Pedido pedido = new Pedido(codigo, fechaHora, codigoCliente);
        double total = 0.0;
        // Construir pedido desde el carrito.
        // NOTA: el stock debe haber sido reservado al agregar al carrito; aquí solo se lee el carrito
        for (int i = 0; i < modeloCarrito.getRowCount(); i++) {
            String codigoProducto = String.valueOf(modeloCarrito.getValueAt(i, 0));
            int cant = 0;
            double precio = 0.0;
            try {
                Object oCant = modeloCarrito.getValueAt(i, 2);
                if (oCant instanceof Number) cant = ((Number) oCant).intValue(); else cant = Integer.parseInt(oCant.toString());
            } catch (Exception ex) {
                cant = 0;
            }
            try {
                Object oPrecio = modeloCarrito.getValueAt(i, 3);
                if (oPrecio instanceof Number) precio = ((Number) oPrecio).doubleValue(); else precio = Double.parseDouble(oPrecio.toString());
            } catch (Exception ex) {
```

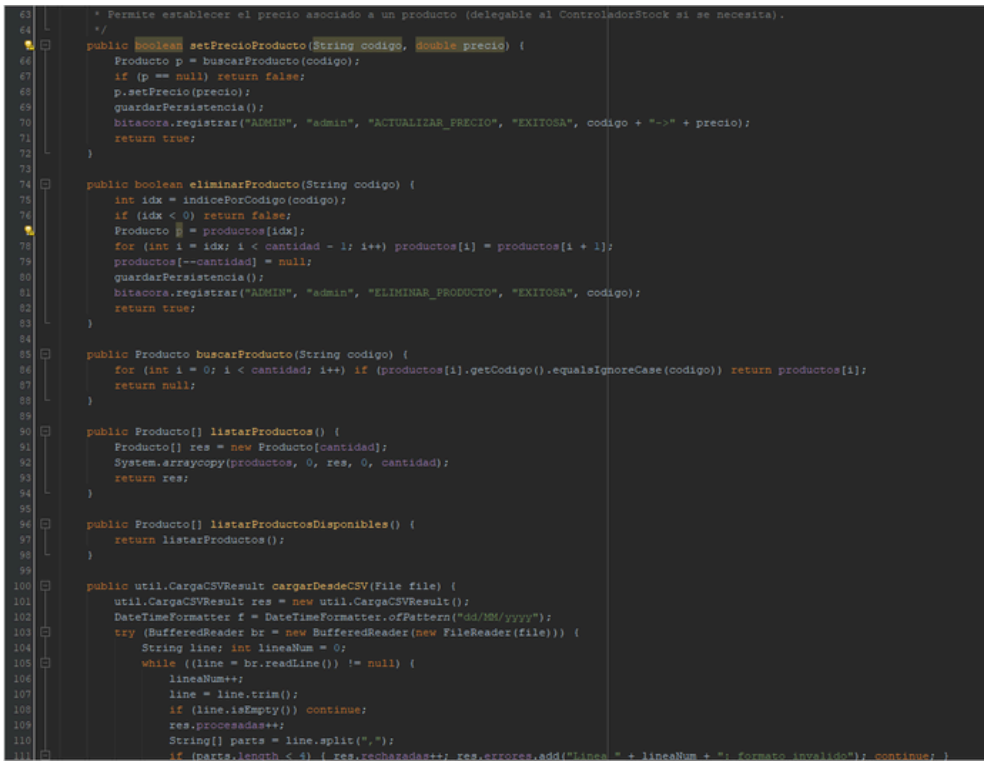
EXPLICACIÓN

En el controlador de productos se utiliza para gestión de productos y stock, con el precio asociados (stock y precios se mantienen en ControladorStock).
En la línea 29 a la 36 verifica si el código ingresado ya esta en la base de datos y si esto no ocurre l producto se crea satisfactoriamente.
En la línea 38 a la 60 busca si existe el producto que buscamos y si existe permite la modificación del atributo por defecto.



```
16  public class ControladorProducto {
17      private Producto[] productos;
18      private int cantidad;
19      private final File persistFile = new File("productos.ser");
20      private final ControladorBitacora bitacora;
21
22      public ControladorProducto(ControladorBitacora bitacora) {
23          this.bitacora = bitacora;
24          this.productos = new Producto[100];
25          this.cantidad = 0;
26          cargarPersistencia();
27      }
28
29      public boolean crearProducto(Producto p) {
30          if (buscarProducto(p.getCodigo()) != null) return false;
31          if (cantidad >= productos.length) productos = redimensionar(productos);
32          productos[cantidad++] = p;
33          guardarPersistencia();
34          bitacora.registrar("ADMIN", "admin", "CREAR_PRODUCTO", "EXITOSA", p.getCodigo());
35          return true;
36      }
37
38      public boolean actualizarProducto(String codigo, String nuevoNombre, String nuevoAtributo) {
39          Producto p = buscarProducto(codigo);
40          if (p == null) return false;
41          p.setNombre(nuevoNombre);
42          // Actualiza atributo según tipo
43          if (p instanceof ProductoTecnologia) {
44              try {
45                  int meses = Integer.parseInt(nuevoAtributo);
46                  ((ProductoTecnologia) p).setMesesGarantia(meses);
47              } catch (NumberFormatException ignored) { }
48          } else if (p instanceof ProductoAlimento) {
49              try {
50                  DateTimeFormatter f = DateTimeFormatter.ofPattern("dd/MM/yyyy");
51                  LocalDate fecha = LocalDate.parse(nuevoAtributo, f);
52                  ((ProductoAlimento) p).setFechaCaducidad(fecha);
53              } catch (Exception ignored) { }
54          } else if (p instanceof ProductoGeneral) {
55              ((ProductoGeneral) p).setMaterial(nuevoAtributo);
56          }
57          guardarPersistencia();
58          bitacora.registrar("ADMIN", "admin", "ACTUALIZAR_PRODUCTO", "EXITOSA", codigo);
59          return true;
60      }
61
62      /**
63       * Permite establecer el precio asociado a un producto (delegable al ControladorStock si se necesita).
64       */
```

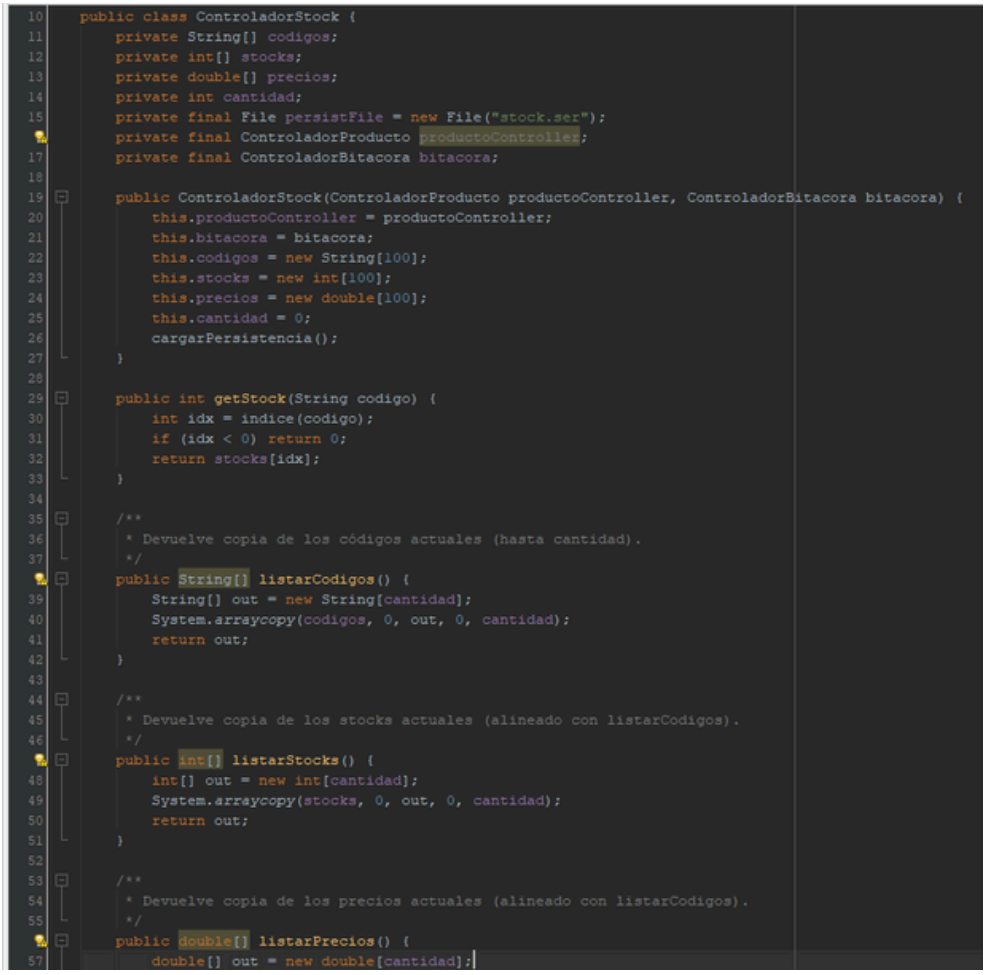
EXPLICACIÓN



```
     * Permite establecer el precio asociado a un producto (delegable al ControladorStock si se necesita).
     */
    public boolean setPrecioProducto(String codigo, double precio) {
        Producto p = buscarProducto(codigo);
        if (p == null) return false;
        p.setPrecio(precio);
        guardarPersistencia();
        bitacora.registrar("ADMIN", "admin", "ACTUALIZAR_PRECIO", "EXITOSA", codigo + "->" + precio);
        return true;
    }

    public boolean eliminarProducto(String codigo) {
        int idx = indicePorCodigo(codigo);
        if (idx < 0) return false;
        Producto p = productos[idx];
        for (int i = idx; i < cantidad - 1; i++) productos[i] = productos[i + 1];
        productos[--cantidad] = null;
        guardarPersistencia();
        bitacora.registrar("ADMIN", "admin", "ELIMINAR_PRODUCTO", "EXITOSA", codigo);
        return true;
    }

    public Producto buscarProducto(String codigo) {
        for (int i = 0; i < cantidad; i++) if (productos[i].getCodigo().equalsIgnoreCase(codigo)) return productos[i];
        return null;
    }

    public Producto[] listarProductos() {
        Producto[] res = new Producto[cantidad];
        System.arraycopy(productos, 0, res, 0, cantidad);
        return res;
    }

    public Producto[] listarProductosDisponibles() {
        return listarProductos();
    }

    public util.CargaCSVResult cargarDesdeCSV(File file) {
        util.CargaCSVResult res = new util.CargaCSVResult();
        DateTimeFormatter f = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            String line; int lineaNum = 0;
            while ((line = br.readLine()) != null) {
                lineaNum++;
                line = line.trim();
                if (line.isEmpty()) continue;
                res.procesadas++;
                String[] parts = line.split(",");
                if (parts.length < 4) { res.rechazadas++; res.errores.add("Linea " + lineaNum + ": formato invalido"); continue; }
```

El código verifica si existe el producto que buscamos y permite la actualización de precios en el mismo, todo para tener un mejor control, lo mismo para eliminar un producto o solo para buscar el producto.
Creo la tabla en la que se almacenaran los datos y también aqui se permiten la carga desde archivos csv.

EXPLICACIÓN



```java
public class ControladorStock {
    private String[] codigos;
    private int[] stocks;
    private double[] precios;
    private int cantidad;
    private final File persistFile = new File("stock.ser");
    private final ControladorProducto productoController;
    private final ControladorBitacora bitacora;

    public ControladorStock(ControladorProducto productoController, ControladorBitacora bitacora) {
        this.productoController = productoController;
        this.bitacora = bitacora;
        this.codigos = new String[100];
        this.stocks = new int[100];
        this.precios = new double[100];
        this.cantidad = 0;
        cargarPersistencia();
    }

    public int getStock(String codigo) {
        int idx = indice(codigo);
        if (idx < 0) return 0;
        return stocks[idx];
    }

    /**
     * Devuelve copia de los códigos actuales (hasta cantidad).
     */
    public String[] listarCodigos() {
        String[] out = new String[cantidad];
        System.arraycopy(codigos, 0, out, 0, cantidad);
        return out;
    }

    /**
     * Devuelve copia de los stocks actuales (alineado con listarCodigos).
     */
    public int[] listarStocks() {
        int[] out = new int[cantidad];
        System.arraycopy(stocks, 0, out, 0, cantidad);
        return out;
    }

    /**
     * Devuelve copia de los precios actuales (alineado con listarCodigos).
     */
    public double[] listarPrecios() {
        double[] out = new double[cantidad];
```

Este es el controlador para stock y precio, en la que los métodos de listar devuelve ya sea el stock, los codigos y los precios para que se puedan representar en la tabla,

EXPLICACIÓN

```java
public class ControladorUsuario {
    private final List<Usuario> usuarios;
    private final ControladorBitacora bitacora;
    private final File persistFile = new File("usuarios.ser");

    public ControladorUsuario(ControladorBitacora bitacora) {
        this.bitacora = bitacora;
        this.usuarios = new ArrayList<>();
        cargarPersistencia();
        asegurarAdminPorDefecto();
    }

    private void asegurarAdminPorDefecto() {
        // Si no existe administrador con código "admin" lo crea con contraseña IPC1F
        if (buscarPorCodigo("admin") == null) {
            Administrador admin = new Administrador("admin", "Administrador", "M", "IPC1F");
            agregarUsuario(admin);
            bitacora.registrar("ADMIN", "admin", "CREAR_USUARIO", "EXITOSA", "Admin por defecto creado");
        }
    }

    public Usuario autenticar(String codigo, String contrasena) {
        Usuario u = buscarPorCodigo(codigo);
        if (u == null) {
            bitacora.registrar("SISTEMA", codigo, "LOGIN", "FALLIDA", "Usuario no encontrado");
            return null;
        }
        if (u.verifyPassword(contrasena)) {
            bitacora.registrar(u.getRol(), u.getCodigo(), "LOGIN", "EXITOSA", "Inicio de sesión correcto");
            return u;
        } else {
            bitacora.registrar(u.getRol(), u.getCodigo(), "LOGIN", "FALLIDA", "Contraseña incorrecta");
            return null;
        }
    }

    public boolean agregarUsuario(Usuario u) {
        if (buscarPorCodigo(u.getCodigo()) != null) return false;
        usuarios.add(u);
        guardarPersistencia();
        bitacora.registrar(u.getRol(), u.getCodigo(), "CREAR_USUARIO", "EXITOSA", "Usuario creado");
        return true;
    }

    public boolean actualizarUsuario(String codigo, String nuevoNombre, String nuevaContrasena) {
        Usuario u = buscarPorCodigo(codigo);
        if (u == null) return false;
        u.setNombre(nuevoNombre);
        u.setPassword(nuevaContrasena);
```

Controlador básico de usuarios usando arrays (vectores) y serialización simple. Archivo persistente: usuarios.ser y el aceptar formato CSV para carga masiva: código,nombre,genero,contrasena.

Permite la creación de usuarios, busqueda de usuarios, actualización de usuarios y eliminacion de los mismos, listar los usuarios, incrementar las ventas y la carga masiva de datos por el csv.

EXPLICACIÓN

```java
package modelo;

public class Administrador extends Usuario {
    private static final long serialVersionUID = 1L;

    public Administrador(String codigo, String nombre, String genero, String contrasena) {
        super(codigo, nombre, genero, contrasena);
    }

    @Override
    public String getRol() {
        return "ADMINISTRADOR";
    }
}
```

```java
public class Bitacora implements Serializable {
    private static final long serialVersionUID = 1L;

    private String timestamp;
    private String tipoUsuario;
    private String codigoUsuario;
    private String operacion;
    private String estado;
    private String descripcion;

    public Bitacora(String tipoUsuario, String codigoUsuario, String operacion, String estado, String descripcion) {
        this.timestamp = LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss"));
        this.tipoUsuario = tipoUsuario;
        this.codigoUsuario = codigoUsuario;
        this.operacion = operacion;
        this.estado = estado;
        this.descripcion = descripcion;
    }

    public String getTimestamp() { return timestamp; }
    public String getTipoUsuario() { return tipoUsuario; }
    public String getCodigoUsuario() { return codigoUsuario; }
    public String getOperacion() { return operacion; }
    public String getEstado() { return estado; }
    public String getDescripcion() { return descripcion; }

    @Override
    public String toString() {
        return String.format("[%s] | %s | %s | %s | %s | %s",
                timestamp, tipoUsuario, codigoUsuario, operacion, estado, descripcion);
    }
}
```

```java
public class Cliente extends Usuario {
    private static final long serialVersionUID = 1L;

    private LocalDate cumpleanos;

    public Cliente(String codigo, String nombre, String genero, LocalDate cumpleanos, String contrasena) {
        super(codigo, nombre, genero, contrasena);
        this.cumpleanos = cumpleanos;
    }

    public LocalDate getCumpleanos() { return cumpleanos; }
    public void setCumpleanos(LocalDate cumpleanos) { this.cumpleanos = cumpleanos; }

    public void setGenero(String genero) { this.genero = genero; }

    @Override
    public String getRol() {
        return "CLIENTE";
    }

    public String getCumpleanosFormato() {
        if (cumpleanos == null) return "";
        return cumpleanos.format(DateTimeFormatter.ofPattern("dd/MM/yyyy"));
    }

    @Override
    public String toString() {
        return String.format("%s - Nac:%s", super.toString(), getCumpleanosFormato());
    }
}
```

En el package modelo se especifican los datos que usaran administrador, cliente y bitacora, el tipo de dato ya sean int, string, double, etc.

EXPLICACIÓN

```java
public abstract class Producto implements Serializable {
    private static final long serialVersionUID = 1L;

    protected String codigo;
    protected String nombre;
    protected String categoria;
    protected double precio = 0.0;

    public Producto(String codigo, String nombre, String categoria) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.categoria = categoria;
    }

    public double getPrecio() { return precio; }
    public void setPrecio(double precio) { this.precio = precio; }

    public String getCodigo() { return codigo; }
    public String getNombre() { return nombre; }
    public String getCategoria() { return categoria; }

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

    public abstract String getDetalle();

    @Override
    public String toString() {
        return String.format("%s - %s [%s]", codigo, nombre, categoria);
    }
}
```

```java
public class ProductoAlimento extends Producto {
    private static final long serialVersionUID = 1L;

    private LocalDate fechaCaducidad;

    public ProductoAlimento(String codigo, String nombre, LocalDate fechaCaducidad) {
        super(codigo, nombre, "Alimento");
        this.fechaCaducidad = fechaCaducidad;
    }

    public LocalDate getFechaCaducidad() { return fechaCaducidad; }
    public void setFechaCaducidad(LocalDate fechaCaducidad) { this.fechaCaducidad = fechaCaducidad; }

    @Override
    public String getDetalle() {
        if (fechaCaducidad == null) return "Fecha de caducidad desconocida";
        DateTimeFormatter f = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        long dias = ChronoUnit.DAYS.between(LocalDate.now(), fechaCaducidad);
        return "Fecha de caducidad: " + fechaCaducidad.format(f) + " | Dias restantes: " + dias;
    }
}
```

```java
public class ProductoGeneral extends Producto {
    private static final long serialVersionUID = 1L;

    private String material;

    public ProductoGeneral(String codigo, String nombre, String material) {
        super(codigo, nombre, "Generales");
        this.material = material;
    }

    public String getMaterial() { return material; }
    public void setMaterial(String material) { this.material = material; }

    @Override
    public String getDetalle() {
        return "Material: " + (material == null ? "" : material);
    }
}
```

En el package modelo se especifican los datos que usaran Producto y las subclases de producto, el tipo de dato ya sean int, string, double, etc.

EXPLICACIÓN

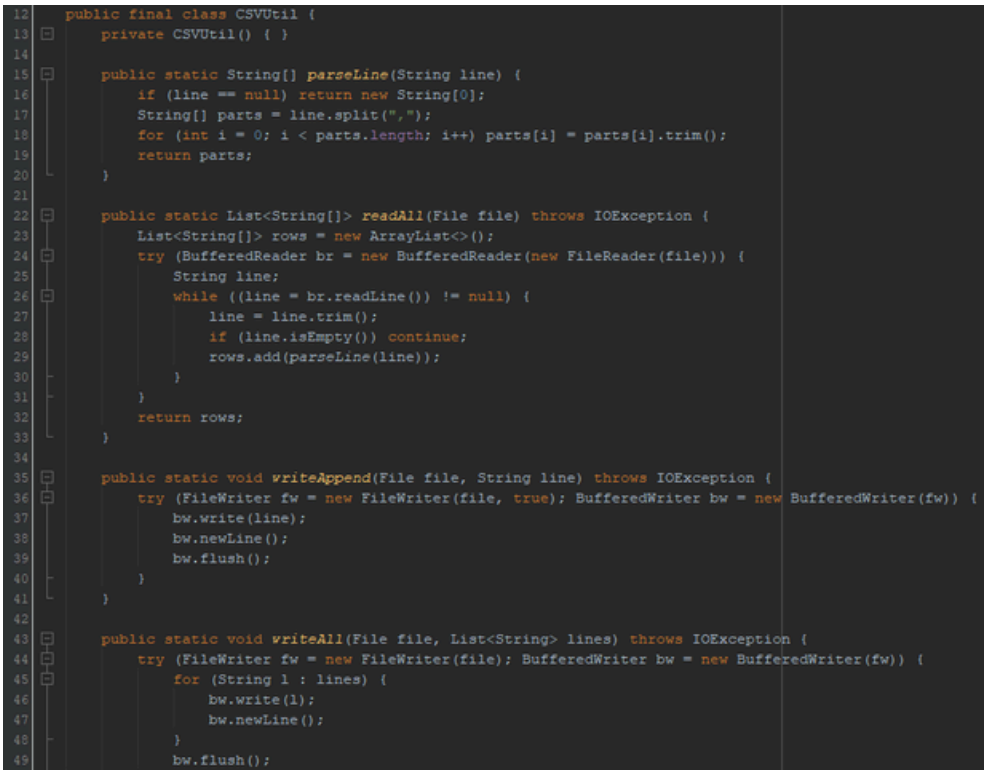
```java
public class ProductoTecnologia extends Producto {
    private static final long serialVersionUID = 1L;

    private int mesesGarantia;

    public ProductoTecnologia(String codigo, String nombre, int mesesGarantia) {
        super(codigo, nombre, "Tecnologia");
        this.mesesGarantia = mesesGarantia;
    }

    public int getMesesGarantia() { return mesesGarantia; }
    public void setMesesGarantia(int mesesGarantia) { this.mesesGarantia = mesesGarantia; }

    @Override
    public String getDetalle() {
        return "Meses de garantia: " + mesesGarantia;
    }
}
```

```java
public abstract class Usuario implements Serializable {
    private static final long serialVersionUID = 1L;

    protected String codigo;
    protected String nombre;
    protected String genero;
    protected String passwordHash; // formato salt:hash (Base64)

    public Usuario(String codigo, String nombre, String genero, String contrasena) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.genero = genero;
        setPassword(contrasena);
    }

    public String getCodigo() { return codigo; }
    public String getNombre() { return nombre; }
    public String getGenero() { return genero; }

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

    public void setGenero(String genero) { this.genero = genero; }

    public void setPassword(String contrasena) {
        if (contrasena == null) this.passwordHash = null; else this.passwordHash = util.PasswordUtil.hashPassword(contrasena);
    }

    public boolean verifyPassword(String contrasena) {
        if (this.passwordHash == null) return false;
        return util.PasswordUtil.verifyPassword(contrasena, this.passwordHash);
    }

    public abstract String getRol();

    @Override
    public String toString() {
        return String.format("%s[%s] - %s", getRol(), codigo, nombre);
    }
}
```

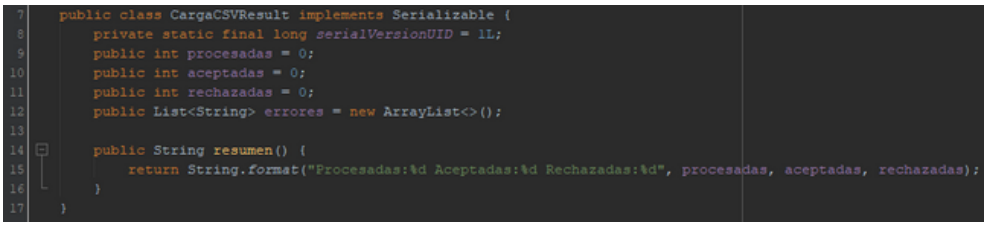
En el package modelo se especifican los datos que usaran Producto y las subclases de producto, asi mismo como las variabls a utilizar de Usuario, el tipo de dato ya sean int, string, double, etc. La verificación de datos creibles y exactos. En el package modelo.Interfaz crea una interfz genérica que define operaciones básicas CRUD. Implementaciones pueden adaptar firmas según necesiten tipos concretos.

```java
package modelo.interfaces;

/**
 * Interfaz genérica que define operaciones básicas CRUD.
 * Implementaciones pueden adaptar firmas según necesiten tipos concretos.
 */
public interface CRUD {
    boolean crear(Object obj);
    boolean actualizar(String codigo, Object... datos);
    boolean eliminar(String codigo);
    Object buscar(String codigo);
    Object[] listar();
}
```

EXPLICACIÓN



```java
public final class CSVUtil {
    private CSVUtil() { }

    public static String[] parseLine(String line) {
        if (line == null) return new String[0];
        String[] parts = line.split(",");
        for (int i = 0; i < parts.length; i++) parts[i] = parts[i].trim();
        return parts;
    }

    public static List<String[]> readAll(File file) throws IOException {
        List<String[]> rows = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            String line;
            while ((line = br.readLine()) != null) {
                line = line.trim();
                if (line.isEmpty()) continue;
                rows.add(parseLine(line));
            }
        }
        return rows;
    }

    public static void writeAppend(File file, String line) throws IOException {
        try (FileWriter fw = new FileWriter(file, true); BufferedWriter bw = new BufferedWriter(fw)) {
            bw.write(line);
            bw.newLine();
            bw.flush();
        }
    }

    public static void writeAll(File file, List<String> lines) throws IOException {
        try (FileWriter fw = new FileWriter(file); BufferedWriter bw = new BufferedWriter(fw)) {
            for (String l : lines) {
                bw.write(l);
                bw.newLine();
            }
            bw.flush();
```



```java
public class CargaCSVResult implements Serializable {
    private static final long serialVersionUID = 1L;
    public int procesadas = 0;
    public int aceptadas = 0;
    public int rechazadas = 0;
    public List<String> errores = new ArrayList<>();

    public String resumen() {
        return String.format("Procesadas:%d Aceptadas:%d Rechazadas:%d", procesadas, aceptadas, rechazadas);
    }
}
```

En el package util ser usa el proceso por carga de csv en los qeu se muestran metodos paar que pueda leer los archivos que se desean y que concuerden con el tipo de datos requeridos.

CONCLUSIÓN

El desarrollo del sistema “Sancarlista Shop” permitió aplicar de manera integral los conocimientos adquiridos en el curso, consolidando los principios de la Programación Orientada a Objetos (POO) y la correcta implementación del patrón arquitectónico Modelo–Vista–Controlador (MVC).

A través de la construcción de una aplicación de escritorio en Java, se logró estructurar un sistema funcional y modular que simula las operaciones fundamentales de una tienda en línea, tales como la gestión de usuarios, productos, inventario y pedidos, integrando además la generación de reportes empresariales en formato PDF mediante la librería iText.

Durante el proceso de desarrollo, se fortalecieron competencias esenciales para la formación del ingeniero en sistemas, como la abstracción, la herencia, el polimorfismo, el encapsulamiento, y la separación de responsabilidades.

este proyecto no solo contribuyó al desarrollo de habilidades técnicas, sino también al fortalecimiento de la lógica de programación, la organización estructural del código y la documentación técnica de un sistema real.

La integración de módulos independientes y comunicados bajo una arquitectura MVC demostró la relevancia de diseñar software mantenible, escalable y adaptable a futuras mejoras.