MANUAL TÉCNICO - SANCARLISTA SHOP

---

1. INFORMACIÓN GENERAL DEL PROYECTO

1.1 Datos del Desarrollo

- **Nombre del Sistema**: Sancarlista Shop
- **Versión**: 1.0
- **Lenguaje de Programación**: Java
- **Patrón Arquitectónico**: MVC (Modelo-Vista-Controlador)
- **Persistencia**: Serialización de objetos + Archivos CSV
- **Interfaz**: Java Swing

1.2 Requisitos del Sistema

Hardware Mínimo:

- Procesador: Intel Core i3 o equivalente
- RAM: 4 GB
- Almacenamiento: 500 MB libres
- Sistema Operativo: Windows 10, Linux, macOS

Software Requerido:

- Java Runtime Environment (JRE) 8 o superior
- Sistema operativo compatible con Java
- Espacio para archivos de datos (.ser, .csv)

---

2. ARQUITECTURA DEL SISTEMA

2.1 Diagrama de Paquetes

proyecto.pkg2.ipc1/

├── Modelo/

├── Vista/

├── Controlador/

├── Reportes_e_hilos/

└── proyecto.pkg2.ipc1/

2.2 Estructura MVC Implementada

Modelo/ (Clases de Entidad y Lógica de Negocio)

├── Usuario.java

├── Administrador.java

├── Vendedor.java

├── Cliente.java

├── Productos.java

├── ProductosTecnologicos.java

├── ProductosAlimenticios.java

├── ProductosGenerales.java

├── Pedidos.java

├── AdministradorUsuarios.java

├── AdministradorProductos.java

├── AdministradorPedidos.java

└── Bitacora.java

Vista/ (Interfaces de Usuario)

├── VistaInicioSesion.java

├── VistaAdministrador.java

├── VistaVendedor.java

├── VistaCliente.java

├── VistaCreacionProductos.java

├── VistaActualizarProducto.java

├── VistaEliminarProducto.java

├── VistaCrearVendedor.java

├── VistaActualizarVendedor.java

├── VistaEliminarVendedor.java

├── VistaAgregarStock.java

├── VistaCrearCliente.java

├── VistaActualizarCliente.java

├── VistaEliminarCliente.java

└── VistaMonitoreoHilos.java

Controlador/ (Lógica de Aplicación)

├── ControladorIniciarSesion.java

├── controladorAdministrador.java

├── ControladorVendedor.java

└── ControladorCliente.java

Reportes_e_hilos/ (Funcionalidades Avanzadas)

├── MonitoreoSistemas.java

└── GeneradorReportes.java

---

3. ESPECIFICACIONES TÉCNICAS DETALLADAS

3.1 Clases del Modelo Principales

3.1.1 Jerarquía de Usuarios

java

```
// Clase base Usuario
public class Usuario implements Serializable {
   private String nombre;
   private String codigo;
   private String contraseña;
   private String genero;
   // Getters y Setters
}

// Herencia aplicada
public class Administrador extends Usuario { }
public class Vendedor extends Usuario {
   private int ventasConfirmadas;
}
public class Cliente extends Usuario {
   private String cumpleañosCliente;
}
```

3.1.2 Jerarquía de Productos

java

```
// Clase base Productos
public class Productos implements Serializable {
   private String nombreProducto;
   private String codigoProducto;
   private String categoria;
   private double precio;
   private int stock;
```

```
}

// Polimorfismo por categorías
public class ProductosTecnologicos extends Productos {
   private int mesesGarantia;
}
public class ProductosAlimenticios extends Productos {
   private String fechaCaducidad;
}
public class ProductosGenerales extends Productos {
   private String materialProducto;
}
```

3.2 Administradores de Datos

3.2.1 AdministradorUsuarios

java

```
public class AdministradorUsuarios implements Serializable {
   private Usuario[] usuarios;
   private int contadorUsuarios;
   private int MAX = 100;

   // Métodos CRUD
   public boolean crearUsuario(Usuario usuario)
   public Usuario buscarUsuarioCodigo(String codigo)
   public boolean eliminarUsuario(String codigo)
   public boolean autenticacion(String codigo, String contraseña)
```

}

3.2.2 AdministradorProductos

java

```
public class AdministradorProductos implements Serializable {
   private Productos[] productos;
   private int contadorProductos;
   private int MAX = 100;

   // Métodos principales
   public boolean crearProductos(Productos producto)
   public Productos buscarProductoCodigo(String codigo)
   public Productos[] obtenerTodosProductos()
   public Productos[] obtenerProductoCategoria(String categoria)
}
```

3.2.3 AdministradorPedidos

java

```
public class AdministradorPedidos implements Serializable {
   private Pedidos[] pedidos;
   private int contadorPedidos;
   private int MAX = 100;

   // Gestión de pedidos
   public boolean crearPedido(Pedidos pedido)
   public boolean confirmarPedido(String codigoPedido)
   public Pedidos[] obtenerPedidosPendientes()
   public String generarCodigoPedido()
```

}

3.3 Controladores Principales

3.3.1 ControladorIniciarSesion

java

```
public class ControladorIniciarSesion {
   public boolean IniciarSesion(String codigo, String contraseña)
   public void redireccionSegunUsuario(Usuario usuario)
   private void crearUsuarioAdminPorDefecto()
}
```

3.3.2 controladorAdministrador

java

```
public class controladorAdministrador {
   // Gestión de productos
   public boolean crearProducto(String nombre, String codigo, String categoria,
                                double precio, String atributoEspecial)
   public boolean actualizarProducto(String codigo, String nuevoNombre, String
nuevoAtributo)

   // Gestión de vendedores
   public boolean crearVendedor(String codigo, String nombre, String genero, String
contraseña)
   public boolean actualizarVendedor(String codigo, String nuevoNombre, String
nuevaContraseña)
}
```

3.3.3 ControladorVendedor

java

```
public class ControladorVendedor {
```

```
    // Gestión de stock
    public boolean agregarStockProducto(String codigoProducto, int cantidad)

    // Gestión de clientes
    public void crearClienteDesdeVentana(VistaCrearCliente ventana)

    // Gestión de pedidos
    public void confirmarPedido(String codigoPedido)
}
```

3.3.4 ControladorCliente

java

```
public class ControladorCliente {
    // Carrito de compras
    public void agregarProductoCarrito(String codigoProducto)
    public void realizarPedido()

    // Gestión de productos
    private void cargarProductosTabla()
}
```

---

4. PERSISTENCIA DE DATOS

4.1 Serialización de Objetos

java

```
// Ejemplo de guardado en archivo .ser
public void guardarEnArchivo() {
    try(ObjectOutputStream salida = new ObjectOutputStream(
```

```
        new FileOutputStream(Archivo_productos))) {

        Productos[] productosGuardar = new Productos[contadorProductos];
        for (int i = 0; i < contadorProductos; i++) {
            productosGuardar[i] = productos[i];
        }
        salida.writeObject(productosGuardar);
        salida.writeInt(contadorProductos);
    } catch (IOException e) {
        // Manejo de errores
    }
}
```

4.2 Archivos de Datos Generados

- usuarios.ser - Datos de todos los usuarios
- productos.ser - Catálogo completo de productos
- pedidos.ser - Historial de pedidos del sistema

4.3 Formatos CSV para Importación

Vendedores:

csv

codigo,nombre,genero,contraseña

V001,Juan Perez,M,password123

Productos:

csv

codigo,nombre,categoria,atributo,precio

P001,Laptop,Tecnología,24,5000.00

Clientes:

csv

codigo,nombre,genero,cumpleaños,contraseña

C001,Carlos Ruiz,M,15/05/1990,cliente123

Stock:

csv

codigo,cantidad

P001,50

P002,100

---

5. ALGORITMOS Y ESTRUCTURAS DE DATOS

5.1 Estructuras Utilizadas

- **Arreglos estáticos**: Para almacenamiento de usuarios, productos, pedidos
- **Búsqueda lineal**: Para localizar elementos por código
- **Algoritmos de ordenamiento**: Implementados manualmente según requerimientos

5.2 Ejemplo de Búsqueda Lineal

java

```
public Productos buscarProductoCodigo(String codigo){
  for (int i = 0; i < contadorProductos; i++){
    if (productos[i].getCodigoProducto().equals(codigo)){
      return productos[i];
    }
  }
  return null;
}
```

5.3 Manejo de Límites

java

```
public boolean crearProductos(Productos producto) {
   if (contadorProductos < MAX &&
      buscarProductoCodigo(producto.getCodigoProducto()) == null){
      // Lógica de inserción
      return true;
   }
   return false;
}
```

---

6. MÉTODOS PRINCIPALES DEL SISTEMA

6.1 Autenticación y Autorización

java

```
// ControladorIniciarSesion.java
public boolean IniciarSesion(String codigo, String contraseña) {
   Usuario usuario = administrador.buscarUsuarioCodigo(codigo);
   if(usuario != null && usuario.getContraseña().equals(contraseña)){
      redireccionSegunUsuario(usuario);
      return true;
   }
   return false;
}
```

6.2 Gestión de Inventario

java

```
// ControladorVendedor.java
public boolean agregarStockProducto(String codigoProducto, int cantidad){
```

```
    Productos producto = adminProductos.buscarProductoCodigo(codigoProducto);
    if (producto != null){
        producto.agregarStock(cantidad);
        adminProductos.guardarEnArchivo();
        return true;
    }
    return false;
}
```

6.3 Procesamiento de Pedidos

```
java
// ControladorCliente.java
public void realizarPedido() {
    if(contadorCarrito == 0){
        JOptionPane.showMessageDialog(vista,"Carrito vacío");
        return;
    }
    // Lógica de creación de pedido
    String codigoPedido = adminPedidos.generarCodigoPedido();
    // ... resto de la implementación
}
```

---

7. CONFIGURACIÓN E INSTALACIÓN

7.1 Requisitos de Compilación

- JDK 8 o superior
- IDE: NetBeans (recomendado) o Eclipse
- Sistema de construcción: Maven (opcional)

7.2 Estructura del Proyecto

text

```
Proyecto2IPC1/
├── src/
│   ├── Modelo/
│   ├── Vista/
│   ├── Controlador/
│   ├── Reportes_e_hilos/
│   └── proyecto/pkg2/ipc1/
├── build/
├── dist/
└── Archivos de datos (.ser)
```

7.3 Generación del Ejecutable

bash

```
# Compilación manual
javac -d build src/**/*.java

# Creación del JAR
jar cfe dist/Proyecto2IPC1.jar proyecto.pkg2.ipc1.Proyecto2IPC1 -C build .
```

---

8. CONSIDERACIONES TÉCNICAS

8.1 Limitaciones Conocidas

- **Límite de elementos**: 100 registros por tipo (configurable)
- **Persistencia**: Requiere permisos de escritura en directorio
- **Concurrencia**: No soporta múltiples usuarios simultáneos

8.2 Manejo de Errores

- Validación de entrada de datos
- Manejo de excepciones de archivos
- Mensajes de error descriptivos
- Recuperación ante fallos de serialización

8.3 Seguridad

- Contraseñas almacenadas en texto plano (mejorable)
- Validación de roles y permisos
- Control de acceso por tipo de usuario

---

9. MANTENIMIENTO Y EXTENSIÓN

9.1 Archivos de Configuración

- Los archivos .ser se regeneran automáticamente si se eliminan
- Estructura de datos compatible con versiones futuras

9.2 Extensiones Posibles

- Implementación de base de datos
- Sistema de logging avanzado
- Reportes adicionales
- Integración con APIs externas