Manual Técnico - Sistema de Gestión de Tienda

Versión: 1.0
Autor: Jimmy Brian Hurtarte López
Carné: 202303768

---

1. Información General del Sistema

1.1 Descripción Detallada

El sistema implementa una solución completa para la gestión de una tienda minorista, con capacidades de:

- Gestión multiusuario con roles específicos
- Inventario categorizado por tipo de producto
- Sistema de ventas con carrito de compras
- Generación automática de reportes
- Monitoreo en tiempo real de actividades
- Importación masiva de datos vía CSV
- Persistencia de datos mediante serialización

1.2 Requerimientos del Sistema

Requerimientos de Hardware Detallados

- Procesador:
    - Mínimo: Intel Core i3/AMD Ryzen 3 2.0 GHz
    - Recomendado: Intel Core i5/AMD Ryzen 5 3.0 GHz o superior
- Memoria RAM:
-
    - Mínimo: 4GB
    - Recomendado: 8GB
    - Óptimo: 16GB para grandes volúmenes de datos
- Almacenamiento:
    - Mínimo: 500MB para la aplicación
    - Recomendado: 2GB para la aplicación y datos
    - Adicional: Espacio para respaldos y reportes
- Pantalla:
    - Resolución mínima: 1280x720

- Resolución recomendada: 1920x1080
- Profundidad de color: 32 bits

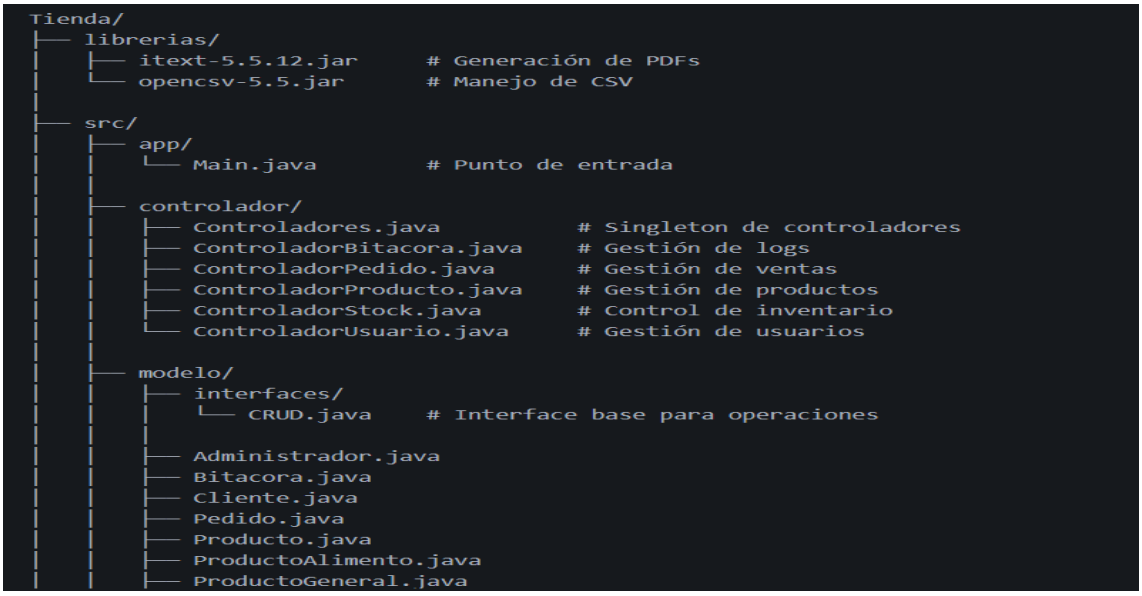
Requerimientos de Software Detallados

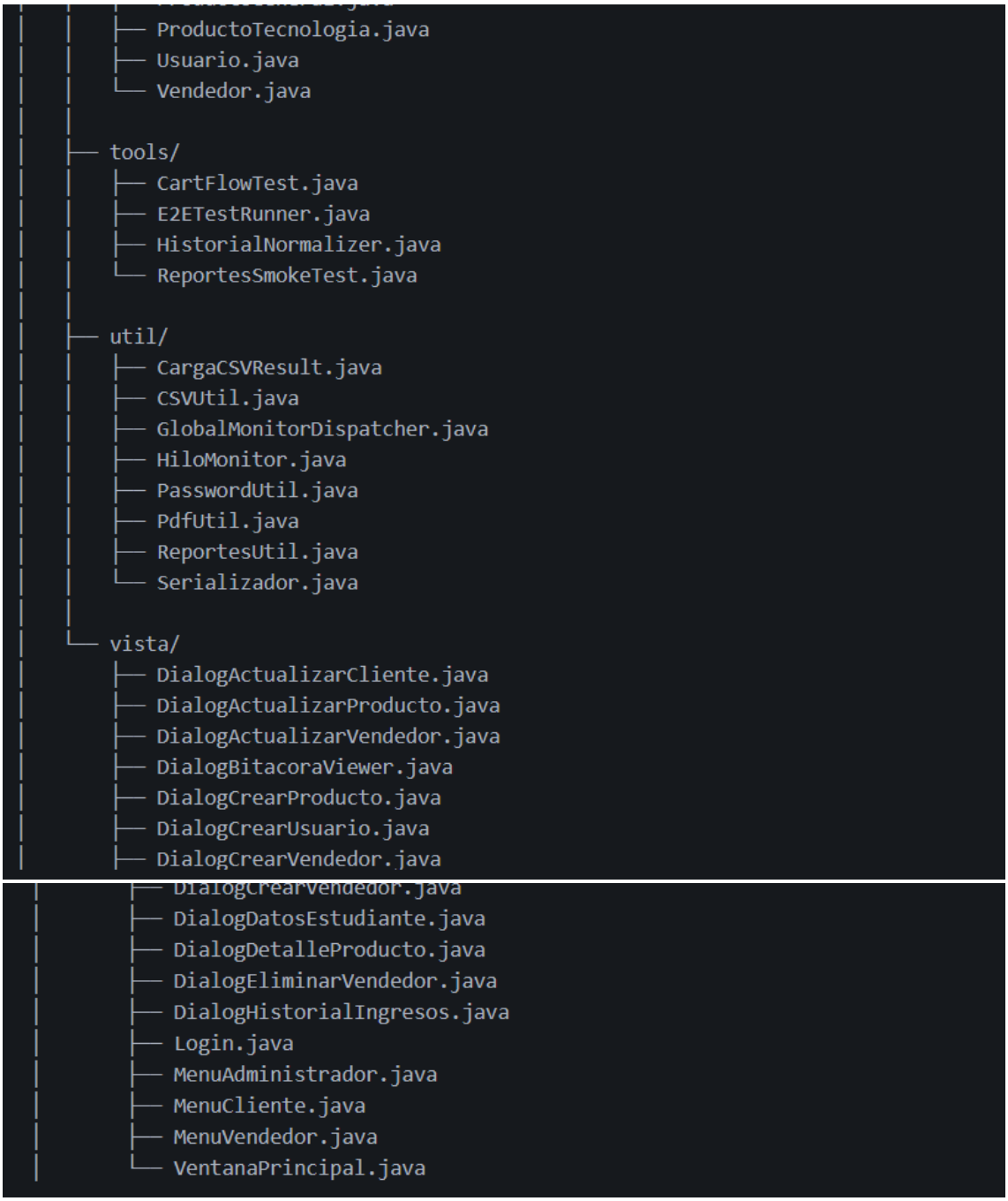
- Sistema Operativo:
  - Windows 10/11 (64 bits)
  - macOS Catalina (10.15) o superior
  - Linux (Ubuntu 20.04 LTS o superior)
- Java:
  - JDK 11 o superior (preferiblemente OpenJDK)
  - Variables de entorno configuradas:
    - JAVA_HOME
    - PATH incluyendo bin del JDK
- IDE Recomendado:
  - NetBeans 12.0 o superior
  - Configuraciones recomendadas:
    - Codificación: UTF-8
    - Formato de línea: LF (Unix) o CRLF (Windows)

---

2. Arquitectura del Sistema

2.1 Estructura Detallada del Proyecto





2.2 Patrones de Diseño Implementados

2.2.1 Patrón MVC

- **Modelo**:
  - Clases en paquete modelo/
  - Encapsulación de datos y lógica de negocio
  - Implementación de interfaces CRUD
- **Vista**:

- Interfaces gráficas en vista/
- Implementación con Swing
- Separación de lógica de presentación

- **Controlador**:
  - Clases en controlador/
  - Mediación entre modelo y vista
  - Gestión de eventos y actualizaciones

2.2.2 Singleton

Implementado en `Controladores.java`:

```java
public class Controladores {
    private static Controladores instance;
    private ControladorUsuario controladorUsuario;
    private ControladorProducto controladorProducto;
    // ...

    private Controladores() {
        inicializarControladores();
    }

    public static Controladores getInstance() {
        if (instance == null) {
            instance = new Controladores();
        }
        return instance;
    }
}
```

2.2.3 Factory Method

Implementado para la creación de productos:

```java
public abstract class Producto {
    public static Producto crearProducto(TipoProducto tipo) {
        switch (tipo) {
            case ALIMENTO:
                return new ProductoAlimento();
            case TECNOLOGIA:
                return new ProductoTecnologia();
            case GENERAL:
                return new ProductoGeneral();
            default:
                throw new IllegalArgumentException("Tipo de producto no válido");
        }
    }
}
```

2.2.4 Observer

Implementado en el sistema de monitoreo:

```
public class GlobalMonitorDispatcher {
    private List<Observer> observers = new ArrayList<>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void notifyEvent(String evento) {
        for (Observer observer : observers) {
            observer.update(evento);
        }
    }
}
```

---

3. Componentes del Sistema

3.1 Módulo de Autenticación

3.1.1 Login.java

```
public class Login extends JFrame {
    private JTextField txtUsuario;
    private JPasswordField txtPassword;
    private ControladorUsuario controlador;

    public Login() {
        initComponents();
        this.controlador = Controladores.getInstance().getControladorUsuario();
    }

    private void btnLoginActionPerformed(ActionEvent evt) {
        String usuario = txtUsuario.getText();
        String password = new String(txtPassword.getPassword());
        Usuario user = controlador.autenticar(usuario, password);
        if (user != null) {
            mostrarMenuSegunRol(user);
        }
    }
}
```

3.1.2 PasswordUtil.java

Implementa encriptación SHA-256:

```
public class PasswordUtil {
    public static String encriptar(String password) {
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hash = digest.digest(password.getBytes(StandardCharsets.UTF_8));
            return Base64.getEncoder().encodeToString(hash);
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("Error al encriptar contraseña", e);
        }
    }
}
```

3.2 Módulo de Usuarios

3.2.1 Jerarquía de Usuarios

```
public abstract class Usuario {
    protected String id;
    protected String nombre;
    protected String password;
    protected TipoUsuario tipo;

    public abstract boolean tienePermiso(String accion);
}

public class Administrador extends Usuario {
    @Override
    public boolean tienePermiso(String accion) {
        return true; // Acceso total
    }
}

public class Cliente extends Usuario {
    private List<Pedido> historialCompras;

    @Override
    public boolean tienePermiso(String accion) {
        return accion.startsWith("COMPRA_") || accion.equals("VER_CATALOGO");
    }
}

public class Vendedor extends Usuario {
    private List<Pedido> ventasRealizadas;
```

3.3 Módulo de Productos (detallado)

3.3.1 Estructura y responsabilidades

- Clase base: Producto
    - Atributos: String id, String nombre, String descripcion, double precio, int stock, String categoria, Map<String,String> metadatos
    - Métodos: getId(), getNombre(), setPrecio(double), ajustarStock(int delta), toCSV(), fromCSV(String[]), validar()
    - Contratos: implementa Serializable y la interfaz CRUD indirectamente a través del ControladorProducto.
- Subclases:
    - ProductoAlimento
        - Atributos adicionales: Date fechaCaducidad, String unidadMedida, double peso
        - Reglas: validación de fechaCaducidad no nula y > fecha actual.
    - ProductoTecnologia
        - Atributos adicionales: String marca, String modelo, String especificaciones
        - Reglas: pueden tener garantía (boolean) y SKU.
    - ProductoGeneral
        - Atributos: String proveedor, String categoriaGeneral

3.3.2 Operaciones importantes (ControladorProducto)

- List<Producto> listarProductos() — devuelve todos o filtra por categoría.
- Producto buscarPorId(String id) — búsqueda rápida por id.
- boolean crearProducto(Producto p) — valida p.validar() y persiste.
- boolean actualizarProducto(Producto p) — reemplaza datos pero preserva id.
- boolean eliminarProducto(String id) — marca como inactivo o remueve.
- boolean ajustarStock(String id, int delta, String motivo) — registra cambio en bitácora y actualiza stock.

3.3.3 Validaciones y reglas de negocio

- Precio >= 0.0.
- Stock >= 0.
- Productos de tipo alimento requieren fecha de caducidad válida.

- Cambio de precio debe registrar auditoría (usuario, fecha, precioAnterior, precioNuevo).

3.3.4 CSV de ejemplo para productos.csv

Formato esperado (encabezado obligatorio):
id,nombre,descripcion,precio,stock,categoria,tipo,fechaCaducidad,marca,modelo,unidadMedida,proveedor
Ejemplo:
P001,Arroz 1kg,Arroz blanco,3.50,120,Alimento,ALIMENTO,2026-05-12,,,,Proveedor A

(Nota: campos no aplicables se dejan vacíos).

3.4 Módulo de Ventas

3.4.1 Pedido — estructura y flujo

- Atributos:
    - String idPedido, String idCliente, String idVendedor, List<ItemPedido> items, double subtotal, double impuesto, double total, Date fecha, EstadoPedido estado
- ItemPedido: String idProducto, int cantidad, double precioUnitario, double subtotalItem.
- Estados: CREADO, PENDIENTE_PAGO, CONFIRMADO, CANCELADO, ENTREGADO.

3.4.2 Flujo de creación de pedido

1. Cliente agrega productos al carrito (en memoria o sesión).
2. Validación de stock disponible por item (llamada a ControladorStock).
3. Cálculo de totals (subtotal, impuesto configurable, descuentos si aplican).
4. Persistencia del pedido en Serializador o estructura de pedidos.
5. Descuento de stock (si política lo requiere al confirmar).
6. Emisión de recibo (PDF por PdfUtil).

3.4.3 Métodos clave (ControladorPedido)

- Pedido crearPedido(Pedido p) — devuelve pedido con id y fecha.
- boolean confirmarPago(String idPedido, MetodoPago m) — valida y cambia estado.
- boolean cancelarPedido(String idPedido, String motivo) — revierte stock si ya había sido descontado.
- List<Pedido> listarPedidosPorCliente(String idCliente, Date desde, Date hasta).

3.4.4 Manejo de pagos (simplificado)

- Integración simulada: la aplicación mantiene un módulo de validación de pago simulado para prácticas (no integra pasarelas externas).
- Seguridad: los datos de tarjeta nunca se almacenan en texto plano; si se llega a integrar un gateway, usar tokenización y HTTPS.

3.5 Sistema de Monitoreo y Bitácora

3.5.1 GlobalMonitorDispatcher y HiloMonitor

- GlobalMonitorDispatcher actúa como hub para eventos relevantes (creación/actualización/eliminación).
- HiloMonitor ejecuta tareas periódicas (sincronización, limpieza de logs, envío de alertas).

3.5.2 Bitácora (Bitacora.java y ControladorBitacora)

- Registro de eventos en bitacora.txt: cada línea con formato legible (fecha|nivel|usuario|componente|evento|detalle).
- Niveles: INFO, WARN, ERROR, SECURITY.
- ControladorBitacora.registrar(String nivel, String usuario, String componente, String evento, String detalle) — función única para evitar duplicidad de formatos.

3.5.3 Ejemplos de eventos registrados

- LOGIN exitoso: 2025-10-26T10:12:11|INFO|admin|Auth|LOGIN|Usuario admin inició sesión.
- STOCK ajustado: 2025-10-26T10:15:00|INFO|vendedor1|Stock|AJUSTE|Producto P001: -2 (venta #F123).

---

4. Gestión de Datos (persistencia, importación, exportación)

4.1 Persistencia

- Mecanismo principal: serialización binaria mediante Serializador.java.
  - Archivos: usuarios.ser, productos.ser (si existe), pedidos.ser (opcional).
  - Ventajas: simple, no requiere DB.
  - Desventajas: escalabilidad limitada y riesgo de incompatibilidad de versiones de clases.
- Alternativa posible: migrar a una base de datos embebida (H2, SQLite) para escalabilidad y consultas.

4.2 Serializador.java (comportamiento)

- Métodos:
  - boolean guardarObjeto(String ruta, Object obj) — escribe con ObjectOutputStream.
  - Object leerObjeto(String ruta) — lee con ObjectInputStream.
  - Manejo de excepciones con reintentos y reporting en bitácora.

4.3 Importación CSV (CSVUtil.java)

- Usa OpenCSV u otra librería para parseo tolerante.

- Funcionalidades:
  - Mapear encabezados a propiedades de objetos.
  - Validación por fila (devuelve CargaCSVResult con filas válidas, inválidas y errores).
  - Soporte de encoding (UTF-8 por defecto).
- Ejemplo de uso:
  - CargaCSVResult<Producto> resultado = CSVUtil.cargarProductos(rutaArchivo);
  - resultado.getFilasInvalidas() contiene mensajes por fila.

4.4 Formatos CSV esperados (resumen)

- clientes.csv: id,nombre,apellido,telefono,correo,direccion
- vendedores.csv: id,nombre,usuario,passwordHash,comision
- stock.csv: idProducto,cantidad,ubicacion
- productos.csv: (ya mostrado en 3.3.4)

---

5. Generación de Reportes (detalle técnico)

5.1 PdfUtil.java — responsabilidades

- Generación de PDFs con iText 5.5.12.
- Reportes soportados:
  - Reporte tabla (tabular datos genéricos).
  - Reporte productos.
  - Reporte pedidos/ventas (filtro fecha, vendedor).

    - Top vendedores.
    - Historial de ingresos/bitácora.
- Estilos: cabecera con logo (si existe), pie de página (fecha, paginado), tablas con encabezado fijo.

5.2 API pública (ejemplos)

- public static boolean generarReporteTabla(String titulo, List<String> columnas, List<List<String>> filas, String rutaSalida)
- public static boolean generarReporteProductos(List<Producto> productos, String rutaSalida)
- public static boolean generarReporteVentas(List<Pedido> pedidos, String rutaSalida)

5.3 Ejemplo de llamada (desde `ControladorReporte` o UI)

```
List<String> columnas = Arrays.asList("ID","Nombre","Precio","Stock");
List<List<String>> filas = productos.stream()
    .map(p -> Arrays.asList(p.getId(), p.getNombre(), String.valueOf(p.getPrecio()), String.valueOf(p.getStock())))
    .collect(Collectors.toList());
PdfUtil.generarReporteTabla("Inventario de Productos", columnas, filas, "reportes/inventario_2025-10-26.pdf");
```

5.4 Manejo de errores y rendimiento

- Generación en hilo separado para UI responsiva.
- Manejo de OOM: paginar la escritura en tablas grandes, liberar referencias a objetos pesados.
- Verificar permisos de escritura y espacio disponible antes de iniciar.

---

6. Interfaces de Usuario (UI) — diseño y comportamiento

6.1 Principios de diseño

- Tecnología: Swing (JFrame, JDialog, JTable, JButton).
- Separación de responsabilidad: la lógica de negocio no debe estar en listeners más allá de orquestación.
- Accesibilidad: etiquetas claras, atajos de teclado (Alt+) y mensajes de validación legibles.

6.2 Ventanas clave y eventos

- VentanaPrincipal: inicializa menús y status bar (usuario conectado, fecha).

- Login: valida credenciales y lanza el Menu correspondiente según rol.
- MenuAdministrador:
    - Acciones: Gestión de usuarios, ver bitácora, importaciones masivas, reportes.
- MenuVendedor:
    - Acciones: Crear pedidos, ver stock, ver ventas propias.
- MenuCliente:
    - Acciones: Ver catálogo, historial de pedidos, perfil.

6.3 Componentes comunes

- JTable con TableModel personalizado para reflejar cambios del modelo.
- Formularios con validación en cada campo (regex para email, rangos para números).
- Diálogos modal para confirmaciones y errores.

6.4 Ejemplo de listener (crear producto)

```
btnGuardar.addActionListener(e -> {
    Producto p = construirProductoDesdeFormulario();
    if (!p.validar()) {
        mostrarError("Campos inválidos");
        return;
    }
    boolean ok = Controladores.getInstance().getControladorProducto().crearProducto(p);
    if (ok) {
        mostrarInfo("Producto creado");
        refrescarTabla();
    } else {
        mostrarError("No se pudo crear el producto (ver bitácora).");
    }
});
```

---

7. Seguridad (ampliado)

7.1 Autenticación y autorización

- Passwords: almacenados como hash SHA-256 con salt por usuario (si PasswordUtil ampliado).
- Roles:
    - ADMIN — permisos totales.
    - VENDEDOR — gestión de ventas y stock.

    - CLIENTE — realizar compras y ver catálogo.
- Autorización: ControladorUsuario.autenticar() retorna Usuario con tipo; la UI habilita o inhabilita acciones en base a tienePermiso(accion).

7.2 Buenas prácticas implementadas

- No almacenar contraseñas en texto plano.
- Registrar intentos fallidos en bitácora con WARN.
- Bloqueo temporal después de N intentos (configurable), p.ej. 5 intentos => bloqueo 15 minutos.
- Sesiones temporales en memoria (si se implementa multi-usuario en red, usar tokens firmados).

7.3 Seguridad de archivos y permisos

- Archivos de datos sensibles (usuarios.ser) con permisos de archivo restringidos (sólo lectura/escritura para el usuario del sistema).
- Recomendación: cifrar archivos críticos o mover a almacenamiento seguro.

---

8. Mantenimiento y respaldo

8.1 Tareas periódicas

- Backup diario de archivos *.ser y bitacora.txt.
- Exportar inventario en CSV semanal.
- Limpieza de registros antiguos en la bitácora (archivar trimestralmente).
- Verificación de integridad de archivos (checksum).

8.2 Procedimiento de backup manual (Windows PowerShell)

- Copiar archivos a carpeta de respaldo con fecha:

```
$fecha = (Get-Date).ToString("yyyy-MM-dd")
Copy-Item -Path "C:\ruta\Proyecto 2\usuarios.ser" -Destination "C:\backups\Proyecto2\usuarios_$fecha.ser"
Copy-Item -Path "C:\ruta\Proyecto 2\bitacora.txt" -Destination "C:\backups\Proyecto2\bitacora_$fecha.txt"
```

8.3 Restauración

1. Detener la aplicación.
2. Reemplazar archivos *.ser en la carpeta de la aplicación con las copias del backup.

3. Reiniciar la aplicación y verificar logs.

8.4 Actualizaciones y migraciones

- Versionado de serialización: usar serialVersionUID y plan de migración si cambian clases serializables.
- Pruebas en entorno staging antes de actualización en producción.

---

9. Despliegue e instalación

9.1 Requisitos previos

- JDK 11+ instalado.
- Variables de entorno JAVA_HOME y PATH correctamente configuradas.

9.2 Instrucciones (Windows)

1. Obtener el ZIP del proyecto y descomprimir en C:\Tienda.
2. Instalar dependencias (si hay jars en librerias/, asegurarse estén presentes).
3. Compilar:

```
cd 'C:\Tienda\Tienda\src'
# Si usa javac directamente (ejemplo): compilar todos los .java (ajustar classpath a librerias)
javac -cp ".;..\\librerias\\itext-5.5.12.jar;..\\librerias\\opencsv-5.5.jar" -d ..\bin ..\app\Main.java
```

4. Ejecutar:

```
cd 'C:\Tienda\Tienda\bin'
java -cp ".;..\\librerias\\itext-5.5.12.jar;..\\librerias\\opencsv-5.5.jar" app.Main
```

(Nota: adaptar rutas y usar un build tool como Maven/Gradle mejora reproducibilidad).

9.3 Despliegue con JAR ejecutable

- Crear manifest.mf con Main-Class y empaquetar con dependencias (fat-jar) o usar one-jar.
- Ejecutar:

```
java -jar Tienda-1.0-all.jar
```

---

10. Pruebas y aseguramiento de calidad

10.1 Estrategia de pruebas

- Unitarias: Pruebas de negocios (validación de producto, cálculo de totales).
- Integración: Serialización/Deserialización, importación CSV → persistencia.
- E2E: Flujos principales (login → crear pedido → generar reporte). Ya hay utilidades en tools/ (E2ETestRunner, CartFlowTest).

10.2 Ejecución de pruebas (sugerencia)

- Si el proyecto usa JUnit, ejecutar mediante el IDE o mvn test/gradle test.
- Si no usa framework, tools/*.java tienen main que simulan flujos; ejecutar en IDE.

10.3 Casos de prueba recomendados

- Crear producto con precio negativo → debe fallar.
- Cargar CSV con filas mal formadas → filas inválidas reportadas.
- Simular múltiples pedidos simultáneos (hilo) → verificar consistencia de stock (si no hay locking, documentar limitación).
- Generación de reportes con 10000 filas — medir memoria y tiempo.

---

11. Contratos y formatos (API interna y estructuras)

11.1 Estructura Pedido (JSON/Serializado)

- JSON ejemplo:


```
{
  "idPedido":"F20251026-001",
  "idCliente":"C001",
  "items":[{"idProducto":"P001","cantidad":2,"precioUnitario":3.5}],
  "subtotal":7.0,
  "impuesto":0.56,
  "total":7.56,
  "fecha":"2025-10-26T10:00:00",
  "estado":"CONFIRMADO"
}
```

11.2 Esquemas CSV (resumen)

- Ya mostrado en 3.3.4 y 4.4. Es importante incluir encabezado y usar UTF-8.

---

12. Configuración y parámetros

12.1 Archivo de configuración recomendado (config.properties)

- Parámetros:
    - impuesto.porcentaje=0.08
    - backup.path=C:\\backups\\Proyecto2
    - login.maxIntentos=5
    - login.bloqueoMinutos=15
    - reportes.outputDir=reportes

12.2 Cargar configuración

- ConfigUtil (sugerido) lee config.properties y provee valores con default.

---

13. Mejores prácticas y mejoras propuestas

13.1 Pequeñas mejoras (bajo riesgo)

- Añadir serialVersionUID consistente para clases serializables.
- Externalizar configuración a config.properties.
- Añadir logs estructurados (JSON) además de bitacora.txt.
- Implementar pruebas unitarias automatizadas (JUnit 5).

13.2 Mejoras de mediano plazo

- Migrar persistencia a base de datos embebida (H2/SQLite).
- Implementar internacionalización (i18n) para cadenas de UI.
- Añadir autenticación con OAuth2 o integración con LDAP (si aplica).

13.3 Refactorings sugeridos

- Extraer lógica de negocio a servicios desacoplados
  (ej. ProductoService, PedidoService).
- Introducir DAO/Repository para acceso a datos.

---

14. Anexos

14.1 Diagramas (texto para importar a StarUML)

- Lista de clases principales (nombre, atributos, métodos) y relaciones (herencia entre Producto y subclases; agregación de Pedido → ItemPedido; asociación Usuario ↔ Pedido):
    - Producto <|-- ProductoAlimento, ProductoTecnologia, ProductoGeneral
    - Usuario <|-- Administrador, Vendedor, Cliente
    - Pedido o-- ItemPedido (composición)
    - Controladores uses ControladorProducto, ControladorUsuario, ControladorPedido, ControladorStock, ControladorBitacora
    - PdfUtil — clase utilitaria estática (no instancia)

(Puedes importar estas relaciones manualmente en StarUML usando clases y flechas de herencia/associación.)

14.2 Ejemplo de CargaCSVResult (estructura)

- Atributos:
    - List<T> objetosValidos
    - List<String> filasInvalidas
    - Map<Integer,String> erroresPorFila

---

15. Diagnóstico y solución de problemas comunes (ampliado)

1. Aplicación no inicia:
    - Verificar versión de JDK y JAVA_HOME.
    - Revisar bitacora.txt y excepciones impresas en consola.
2. Error al leer usuarios.ser:
    - Posible corrupción: recuperar desde backup o descartar y recrear usuario admin.
    - Revisar serialVersionUID.
3. CSV cargado con caracteres raros:
    - Confirmar encoding (usar UTF-8).
    - Verificar separador; si usuario usa ;, permitir opción de separador.
4. Problemas de memoria en reportes grandes:
    - Generar reporte por streaming (iText permite escribir por partes).
    - Aumentar heap JVM con -Xmx si es necesario.