

# MANUAL TÉCNICO

## dojo de karate "Cobra"

Curso

IPC1

Carné

201907179

Nombre

Wilson Manuel Santos Ajcot

# I. TABLA DE CONTENIDO

II. INTRODUCCIÓN	3
III. OBJETIVOS	3
IV. ESPECIFICACIÓN TÉCNICA	4
• REQUISITOS DEL HARDWARE	4
• REQUISITOS DEL SOFTWARE	4
V. Dirigido	3
VI. LÓGICA DEL PROGRAMA	4
VII. DIAGRAMAS DE FLUJO	16
• Login	16
• Menú principal	17
• Agregar producto	18
• Agregar producto (masiva)	19
• Agregar Producto a Inventario	20
• Realizar venta	21
• Registrar Venta	22
• Generar Factura	23
• Ver Inventario	24
• Reporte Top 5	25
• Reporte Histórico	26

## II. INTRODUCCIÓN

El "Código Cobra: Sistema de Gestión de Inventarios y Ventas" es una herramienta diseñada para facilitar la administración de inventarios, el proceso de ventas y la generación de reportes en cualquier negocio. Con funciones para agregar productos, realizar ventas, cargar datos de forma masiva desde archivos CSV y visualizar reportes de productos vendidos e historial de ventas, este sistema simplifica la gestión diaria de recursos.

Inspirado en el espíritu de Cobra Kai y con un enfoque práctico, el sistema es intuitivo, rápido y adaptable, brindando información clave para la toma de decisiones en el ámbito de ventas y control de inventarios.

## III. OBJETIVOS

El objetivo principal de este manual es proporcionar a los programadores y estudiantes una comprensión detallada del desarrollo de la aplicación, enfocándose en la implementación de un sistema para la gestión de usuarios, inventarios, ventas y reportes.

El sistema está diseñado para que los usuarios puedan interactuar con la aplicación de forma sencilla, y los programadores puedan aprender a aplicar los principios de la programación en Java para crear aplicaciones funcionales.

Además, el manual ofrece una guía para el uso adecuado de validaciones y manejo de archivos, fundamentales para el desarrollo de este tipo de sistemas.

## IV. DIRIGIDO

Este manual está orientado a todas aquellas personas interesadas en entender el funcionamiento y desarrollo de esta aplicación, así como a programadores que deseen utilizar esta práctica como referencia para aplicar métodos y funciones en sus propios proyectos.

El manual proporciona una guía clara sobre cómo se implementaron las diversas funcionalidades del sistema, permitiendo a los usuarios comprender y adaptar las soluciones propuestas.

## V. ESPECIFICACIÓN TÉCNICA

### • REQUISITOS DEL HARDWARE

- Computadora de escritorio o portátil.
- Memoria RAM: Mínimo 4 GB.
- Almacenamiento: Disco duro de 250 GB o superior.
- Procesador Intel Core i3 o superior
- Resolución Gráfica: Resolución mínima de 1024 x 768 píxeles

### • REQUISITOS DEL SOFTWARE

- Sistema Operativo: Windows 7 o superior.
- Java Runtime Environment (JRE): Versión 8.2 o superior.
- Entorno de desarrollo recomendado: IntelliJ IDEA, Eclipse o cualquier IDE compatible con Java.
- Librería javax.SWING.
- Librería IO
- Librería java.util

## VI. LÓGICA DEL PROGRAMA

Funcionalidad	Descripción	Métodos/Funciones Contenidas.
Autenticación	Contiene la lógica que gestiona el login de los usuarios y la validación de sus credenciales.	Login
Menú Principal	Muestra el menú principal donde el usuario puede elegir diferentes opciones (agregar producto, ventas, etc.).	menu
Agregar Producto	Contiene la lógica que gestiona el ingreso de producto desde consola y valida que los datos sean correctos.	agregarProducto

<b>Carga Masiva</b>	Contiene la lógica que gestiona la carga masiva de productos desde archivos y valida que los datos sean correctos.	cargaMasiva
<b>Agregar Producto al Inventario</b>	Contiene la lógica que valida y agrega productos al inventario de manera controlada.	agregarProductoInventario
<b>Ventas</b>	Contiene la lógica que gestiona el proceso de ventas, validando la cantidad de productos y calculando el total de la venta.	realizarVenta
<b>Realizar Venta</b>	Registra los detalles de la venta realizada en el sistema.	registrarVenta
<b>Inventario</b>	Contiene la lógica relacionada con visualización de los productos disponibles en el inventario.	verInventario
<b>Reportes</b>	Genera reportes como el de productos más vendidos y el historial de ventas.	ReporteTop, ReporteHistorico
<b>Generar Factura</b>	Contiene la lógica que genera una factura con los detalles de los productos vendidos.	generarFactura
<b>Validaciones</b>	Contiene las funciones que validan la existencia de productos, la validez del precio y otros.	ExistenProducto, PrecioValido,

- **Clases principales**

La clase principal contiene métodos estáticos que implementan toda la lógica del sistema

- **Método Main**

Este método es el encargado de llamar el método login.

```
public static void main(String[] args) {
    login();
}
```

- **Método Login**

El método login solicita las credenciales del usuario en un ciclo. Si las credenciales son correctas, muestra un mensaje de bienvenida y cambia el estado de logged a true. Si son incorrectas, pide los datos nuevamente. Al iniciar sesión correctamente, se llama al método menu.

```
public static void login() {
    System.out.println("\n*****");
    System.out.println("*          LOGIN          *");
    System.out.println("*****");

    do {
        System.out.println("\nIngrese sus credenciales:");
        System.out.print("Usuario: ");
        inUser = loginSc.nextLine();
        System.out.print("Contraseña: ");
        inPass = loginSc.nextLine();

        if (Objects.equals(inUser, user) && Objects.equals(inPass, password)) {
            System.out.println("\n*****");
            System.out.println("*          ¡BIENVENIDO ADMINISTRADOR!          *");
            System.out.println("*****");
            logged = true;
        } else {
            System.out.println("\n-----");
            System.out.println("          Usuario o contraseña incorrectos.          ");
            System.out.println("          Por favor, inténtelo de nuevo.          ");
            System.out.println("-----");
        }
    } while (!logged);

    menu(); // Llama al menú después de iniciar sesión correctamente
}
```

- **Método menú**

El método menu despliega un menú interactivo para el usuario, donde se valida la opción seleccionada, asegurando que esté dentro del rango permitido. Si la opción es válida, se ejecuta la acción correspondiente utilizando un switch. En caso de que la entrada no sea

válida, se maneja la excepción y se solicita una nueva opción. El ciclo continúa hasta que el usuario selecciona la opción de salida. Al seleccionar "Salir", se cierra el sistema usando un método de terminación controlada.

```
public static void menu() {
    do {
        System.out.println("\n");
        System.out.println("*****");
        System.out.println("MENU PRINCIPAL");
        System.out.println("*****");
        System.out.println(" 1. Agregar un producto");
        System.out.println(" 2. Cargar productos (Masivo)");
        System.out.println(" 3. Ver inventario");
        System.out.println(" 4. Realizar una venta");
        System.out.println(" 5. Reporte (ap 5 Productos mas vendidos)");
        System.out.println(" 6. Reporte histórico de Ventas");
        System.out.println(" 7. Salir");
        System.out.print("Seleccione una opción: ");

        try {
            menuOption = menuSc.nextLine(); // Leer esta la opción seleccionada
            menuSc.nextLine(); // Limpiar el buffer

            // Verificar si la opción está dentro del rango permitido
            if (menuOption < 1 || menuOption > 7) {
                System.out.println("\n");
                System.out.println("HA INGRESADO UNA OPCIÓN INVÁLIDA.");
                System.out.println("Por favor, seleccione una opción válida.");
                continue; // Volver a mostrar el menú
            }

            // Ejecutar la acción según la opción seleccionada
            switch (menuOption) {
                case 1:
                    agregarProducto();
                    break;
                case 2:
                    cargaMasiva();
                    break;
                case 3:
                    verInventario();
                    break;
                case 4:
                    realizarVenta();
                    break;
                case 5:
                    reporteTop();
                    break;
                case 6:
                    reporteHistorico();
                    break;
                case 7:
                    System.out.println("Saliendo del sistema...");
                    System.out.println("¡¡¡QUIVA PRONTO, ADMINISTRADOR!!");
                    System.out.println("¡Gracias por usar el sistema!");
                    System.out.println("Cerrando la aplicación...");
                    System.exit(0); // Termina el programa
                    break;
                default:
                    // Este caso nunca ocurrirá debido a la validación previa
                    System.out.println("Opción inválida. Intente de nuevo.");
            }
        } catch (Exception e) {
            // Si ocurre una excepción (por ejemplo, ingreso de letras o caracteres no
            // válidos)
            System.out.println("\n");
            System.out.println("HA INGRESADO UNA OPCIÓN INVÁLIDA.");
            System.out.println("Por favor, seleccione una opción válida.");
            menuSc.nextLine(); // Limpiar el buffer para evitar un bucle infinito
        }
    } while (menuOption != 7); // Repite si la opción no es salir
}
```



## • Método agregarProducto

El método agregarProducto valida que el inventario no esté lleno, que el producto no exista y que el precio sea válido. Si todo es correcto, agrega el producto al inventario. Si hay errores, muestra mensajes de advertencia o captura excepciones.

```
public static void agregarProducto() {
    try {
        if (Producto.finder().size() > 100) {
            System.out.println("INVENTARIO LLENO.");
            System.out.println("No se pueden agregar más productos.");
            return;
        }

        System.out.println("Ingrese el código del producto: ");
        String codigo = agregarProducto().codigo();

        if (Producto.finder().size() > 100) {
            System.out.println("No se pueden agregar más productos.");
            return;
        }

        double precio = null;
        boolean precioValido = false;

        while (!precioValido) {
            try {
                System.out.println("Ingrese el precio del producto: ");
                precio = agregarProducto().precio();
                agregarProducto().precioValido();
                precioValido = true;
            } catch (Exception e) {
                System.out.println("El precio debe ser mayor a 0. Intente de nuevo.");
                precioValido = false;
            }
        }

        agregarProducto().precioValido();
        agregarProducto().codigoValido();
        agregarProducto().precioValido();
    } catch (Exception e) {
        System.out.println("Se produjo un error. Mensaje: " + e.getMessage());
        e.printStackTrace();
    }
}
```

## • Método cargaMasiva

El método cargaMasiva permite cargar productos desde un archivo CSV. Abre un cuadro de diálogo para seleccionar el archivo, lee las líneas y procesa cada producto, validando que el precio sea mayor a 0 y que el producto no exista en el inventario. Si los datos son válidos, los agrega al inventario. Si ocurre un error al leer el archivo o procesar los datos, muestra mensajes de error correspondientes.



```

public static void main(String[] args) {
    // Crear un JFrame para almacenar el producto a agregar
    JFrame frame = new JFrame();
    frame.setSize(400, 300);
    frame.setVisible(true);

    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setSelectedFile(new File("archivo.txt"));

    int result = fileChooser.showOpenDialog(frame); // Usar estas cosas como componentes padre

    if (result == JFileChooser.APPROVE_OPTION) {
        String filePath = fileChooser.getSelectedFile().getAbsolutePath();

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            String line;
            boolean isFirstLine = true;

            while ((line = br.readLine()) != null) {
                if (isFirstLine) {
                    isFirstLine = false; // Se han encadenado
                    continue;
                }

                String[] parts = line.split(",");
                if (parts.length == 3) {
                    String nombre = parts[0];
                    try {
                        double precio = Double.parseDouble(parts[1]);

                        // Validar precio
                        if (precio <= 0) {
                            System.out.println("El precio debe ser mayor a 0 en la línea: " + line);
                            continue;
                        }

                        // Validar si ya existe
                        if (existeProducto(nombre)) {
                            System.out.println("El producto ya existe en el inventario.");
                            continue;
                        }

                        agregarProductoInventario(nombre, precio);
                    } catch (NumberFormatException e) {
                        System.out.println("Error de formato en la línea: " + line);
                    }
                }
            }
        } catch (IOException e) {
            System.err.println("Error al leer el archivo: " + e.getMessage());
        }
    } else {
        System.out.println("Operación cancelada.");
    }

    frame.dispose(); // Cerrar el JFrame después de usarlo
}

```

## • Menú agregarProductoInventario

El método `agregarProductoInventario` agrega un producto al inventario. Si el inventario está lleno, muestra un mensaje y no realiza ninguna acción. Si hay espacio, guarda el nombre, precio y cantidad inicial del producto en la matriz `ProductsMatrix` y aumenta el contador `Product_Counter`. Finalmente, muestra un mensaje confirmando la adición del producto.

```

public static void agregarProducto(int cantidad, String nombre, double precio) {
    if (Producto_Existen == Men_Productos) {
        System.out.println("No se pueden agregar más productos.");
        return;
    }

    ProductsMatrix[Product_Counter][0] = nombre;
    ProductsMatrix[Product_Counter][1] = Double.parseDouble(precio);
    ProductsMatrix[Product_Counter][2] = cantidad;
    Producto_Existen = true;
    System.out.println("Producto agregado correctamente.");
}

```

- ### Método existeProducto

El método existeProducto verifica si un producto ya está en el inventario. Recorre la matriz ProductsMatrix buscando un producto cuyo nombre coincida con el proporcionado (ignorando mayúsculas y minúsculas). Si encuentra una coincidencia, devuelve true; si no, retorna false.

```

public static boolean existeProducto(String nombre) {
    for (int i = 0; i < Product_Counter; i++) {
        if (ProductsMatrix[i][0].equalsIgnoreCase(nombre)) {
            return true; // El producto ya existe
        }
    }
    return false; // El producto no existe
}

```

- ### Método verInventario

El método verInventario muestra todos los productos en el inventario. Recorre la matriz ProductsMatrix e imprime el nombre, precio y cantidad vendida de cada producto, junto con un número de índice para cada uno.

```

public static void verInventario() {
    System.out.println("Inventario de productos:");

    for (int i = 0; i < Product_Counter; i++) {
        System.out.println(
            "i = " + i + ", Nombre: " + ProductsMatrix[i][0] + ", Precio: " + ProductsMatrix[i][1] + " ----- Cantidad: " + ProductsMatrix[i][2]);
    }
}

```

- ### Método Realizar venta

El método realizarVenta gestiona el proceso de ventas, verificando primero si hay productos disponibles en el inventario. Luego, solicita los datos del cliente y permite seleccionar productos para agregar a la compra, validando la selección y la cantidad. Calcula el total de la venta, actualiza el inventario y registra los detalles de cada producto comprado. Una vez que el usuario finaliza la compra, se registra la venta y se genera la factura correspondiente con los detalles de los productos y el total.



- **Método registrarVenta**

El método registrarVenta guarda los detalles de una venta en los arreglos correspondientes (clientes, NIT, totales y detalles de ventas), siempre que no se haya alcanzado el límite de ventas (MAX\_VENTAS). Si el número de ventas es menor que el límite, registra la venta y aumenta el contador; de lo contrario, muestra un mensaje indicando que no se pueden registrar más ventas.

```
private static void registrarVenta(String cliente, String nit, double total, String detalle) {
    if (contadorVentas < MAX_VENTAS) {
        clientes[contadorVentas] = cliente;
        nits[contadorVentas] = nit;
        totalesVentas[contadorVentas] = total;
        detallesVentas[contadorVentas] = detalle;
        contadorVen++;
        System.out.printf(" Venta registrada exitosamente. Total: Qs.2fen", total);
    } else {
        System.out.println(" No se pueden registrar mas ventas, Limite alcanzado.");
    }
}
```

- **Método generarFactura**

El método generarFactura crea una factura en formato HTML para una venta. Utiliza el nombre del cliente y el NIT para generar un archivo único. El contenido HTML incluye el nombre y NIT del cliente, los detalles de los productos comprados (producto, cantidad, precio y total), y el total de la compra. Si se encuentra algún error al procesar los detalles de la compra, lo maneja y muestra un mensaje de error. El archivo se guarda con un nombre basado en el cliente y NIT.

```

public static void generarReporteTop5ProductosDisponibles(Sistema de Ventas s, String fecha) throws IOException {
    String nombreReporte = "Reporte de Ventas de los 5 productos más vendidos disponibles";
    try {
        // Verificar si hay productos disponibles
        ArrayList<Producto> productosDisponibles = s.getProductosDisponibles();
        if (productosDisponibles.isEmpty()) {
            throw new IOException("No hay productos disponibles para generar el reporte.");
        }
        // Ordenar los productos por cantidad vendida
        ArrayList<Producto> productosOrdenados = s.getProductosOrdenados();
        // Generar el reporte HTML
        String html = "<h1>Reporte de Ventas de los 5 productos más vendidos disponibles</h1>";
        html += "<table>";
        for (Producto p : productosOrdenados) {
            html += "<tr>";
            html += "<td>";
            html += p.getId();
            html += "</td>";
            html += "<td>";
            html += p.getNombre();
            html += "</td>";
            html += "<td>";
            html += p.getPrecio();
            html += "</td>";
            html += "<td>";
            html += p.getCantidadVendida();
            html += "</td>";
            html += "</tr>";
        }
        html += "</table>";
        // Guardar el reporte en un archivo
        FileWriter fw = new FileWriter("Reporte de Ventas de los 5 productos más vendidos disponibles.html");
        fw.write(html);
        fw.close();
    } catch (IOException e) {
        throw new IOException("Error al generar el reporte de ventas de los 5 productos más vendidos disponibles.");
    }
}

// Método para generar el reporte de ventas de los 5 productos más vendidos
public static void generarReporteTop5ProductosVendidos(Sistema de Ventas s, String fecha) throws IOException {
    String nombreReporte = "Reporte de Ventas de los 5 productos más vendidos";
    try {
        // Verificar si hay productos disponibles
        ArrayList<Producto> productosDisponibles = s.getProductosDisponibles();
        if (productosDisponibles.isEmpty()) {
            throw new IOException("No hay productos disponibles para generar el reporte.");
        }
        // Ordenar los productos por cantidad vendida
        ArrayList<Producto> productosOrdenados = s.getProductosOrdenados();
        // Generar el reporte HTML
        String html = "<h1>Reporte de Ventas de los 5 productos más vendidos</h1>";
        html += "<table>";
        for (Producto p : productosOrdenados) {
            html += "<tr>";
            html += "<td>";
            html += p.getId();
            html += "</td>";
            html += "<td>";
            html += p.getNombre();
            html += "</td>";
            html += "<td>";
            html += p.getPrecio();
            html += "</td>";
            html += "<td>";
            html += p.getCantidadVendida();
            html += "</td>";
            html += "</tr>";
        }
        html += "</table>";
        // Guardar el reporte en un archivo
        FileWriter fw = new FileWriter("Reporte de Ventas de los 5 productos más vendidos.html");
        fw.write(html);
        fw.close();
    } catch (IOException e) {
        throw new IOException("Error al generar el reporte de ventas de los 5 productos más vendidos.");
    }
}

// Método para generar el reporte de ventas de los 5 productos más vendidos
public static void generarReporteTop5ProductosVendidos(Sistema de Ventas s, String fecha) throws IOException {
    String nombreReporte = "Reporte de Ventas de los 5 productos más vendidos";
    try {
        // Verificar si hay productos disponibles
        ArrayList<Producto> productosDisponibles = s.getProductosDisponibles();
        if (productosDisponibles.isEmpty()) {
            throw new IOException("No hay productos disponibles para generar el reporte.");
        }
        // Ordenar los productos por cantidad vendida
        ArrayList<Producto> productosOrdenados = s.getProductosOrdenados();
        // Generar el reporte HTML
        String html = "<h1>Reporte de Ventas de los 5 productos más vendidos</h1>";
        html += "<table>";
        for (Producto p : productosOrdenados) {
            html += "<tr>";
            html += "<td>";
            html += p.getId();
            html += "</td>";
            html += "<td>";
            html += p.getNombre();
            html += "</td>";
            html += "<td>";
            html += p.getPrecio();
            html += "</td>";
            html += "<td>";
            html += p.getCantidadVendida();
            html += "</td>";
            html += "</tr>";
        }
        html += "</table>";
        // Guardar el reporte en un archivo
        FileWriter fw = new FileWriter("Reporte de Ventas de los 5 productos más vendidos.html");
        fw.write(html);
        fw.close();
    } catch (IOException e) {
        throw new IOException("Error al generar el reporte de ventas de los 5 productos más vendidos.");
    }
}

```

## • Método reporteTop

El método `reporteTop()` genera un reporte HTML con los 5 productos más vendidos del inventario. Primero verifica si hay productos disponibles y luego recopila sus nombres, precios y cantidades en tres arreglos. A continuación, ordena los productos de acuerdo con la cantidad vendida en orden descendente utilizando un algoritmo de ordenación simple. Después, crea un archivo HTML que muestra los 5 productos más vendidos, su precio y cantidad, mostrando menos si hay menos de 5 productos. Si ocurre un error al generar el archivo, se muestra un mensaje de error, y se asegura de manejar excepciones durante todo el proceso.





## • Método reporteHistorico

El método `reporteHistorico()` genera un archivo HTML que muestra un reporte con el historial de todas las ventas registradas. Primero verifica si hay ventas realizadas; si no, imprime un mensaje indicando que no hay ventas registradas. Luego, crea un archivo HTML donde escribe los detalles de cada venta, incluyendo el nombre del cliente, el NIT, el total de la venta y el detalle de los productos vendidos (con saltos de línea formateados como `<br>`). Si ocurre algún error durante la creación del archivo, se maneja mediante una excepción. Finalmente, muestra un mensaje indicando si el reporte fue generado exitosamente o si hubo un error.

```
public static void reporteHistorico() {
    if (consultarVentas() == 0) {
        System.out.println("No hay ventas registradas.");
        return;
    }

    try {
        FileWriter writer = new FileWriter("reporte historico.html");
        writer.write("<html><head><title>Reporte Historico de Ventas</title>");
        writer.write("<style>");
        writer.write("body { font-family: Arial, sans-serif; margin: 0; padding: 0; background-color: #f2f2f2; }");
        writer.write("h1 { background-color: #000000; color: white; padding: 10px; margin: 0; }");
        writer.write("table { width: 100%; border-collapse: collapse; margin-top: 20px; background-color: white; }");
        writer.write("tr, td { border: 1px solid #ccc; padding: 5px; border-bottom: 1px solid #ddd; }");
        writer.write("tr { background-color: #f2f2f2; }");
        writer.write("</style></head><body>");
        writer.write("<h1><h1>Reporte de Ventas</h1>");
        writer.write("<div class='container'>");
        writer.write("<table><thead><tr><th>NIT</th><th>Nombre</th><th>Total</th><th>Detalle</th></tr></thead>");

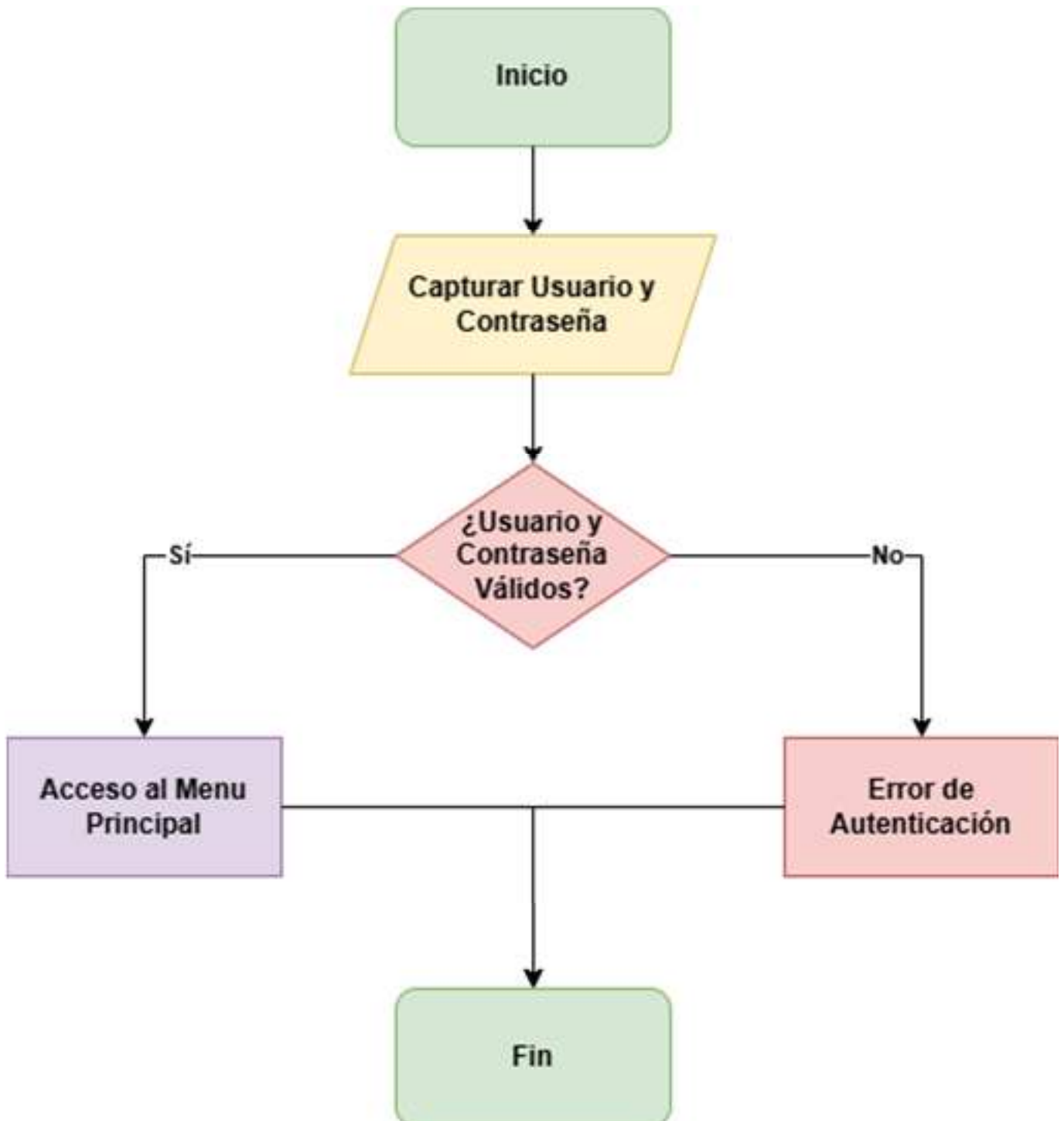
        for (int i = 0; i < consultarVentas(); i++) {
            writer.write("
                <tr><td>"+consultarVentas(i).nit+"</td>";
            writer.write("
                <td>"+consultarVentas(i).cliente+"</td>";
            writer.write("
                <td>"+consultarVentas(i).total+"</td>";
            writer.write("
                <td>"+consultarVentas(i).detalle+"</td>";
            writer.write("
            </tr>");
        }

        writer.write("</table></div></body></html>");
        System.out.println("Reporte historico generado exitosamente. Se creó el archivo: "+ "reporte historico.html");
    } catch (IOException e) {
        System.out.println("Error al generar el historico: " + e.getMessage());
    }
}
```

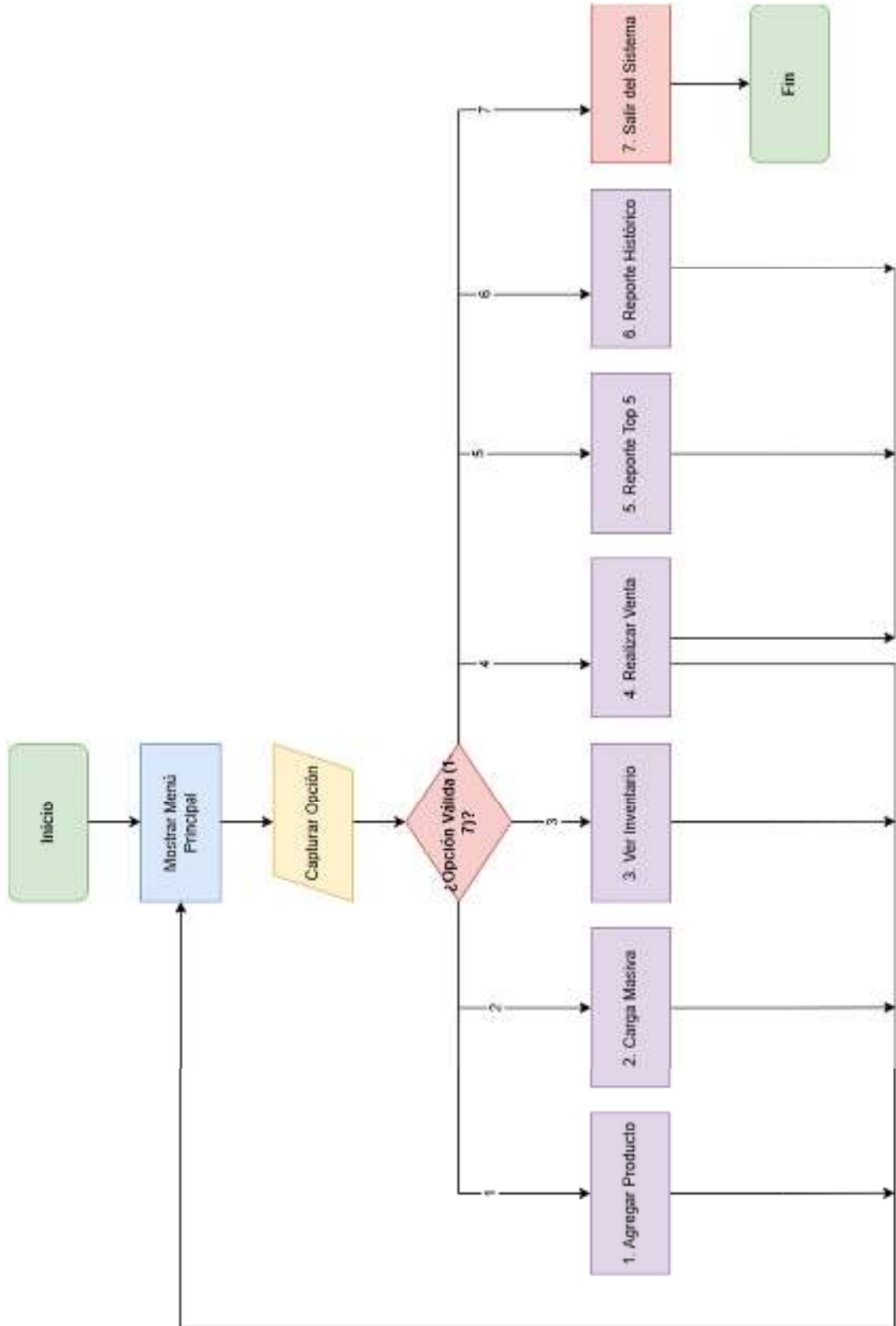


## VII. DIAGRAMAS DE FLUJO

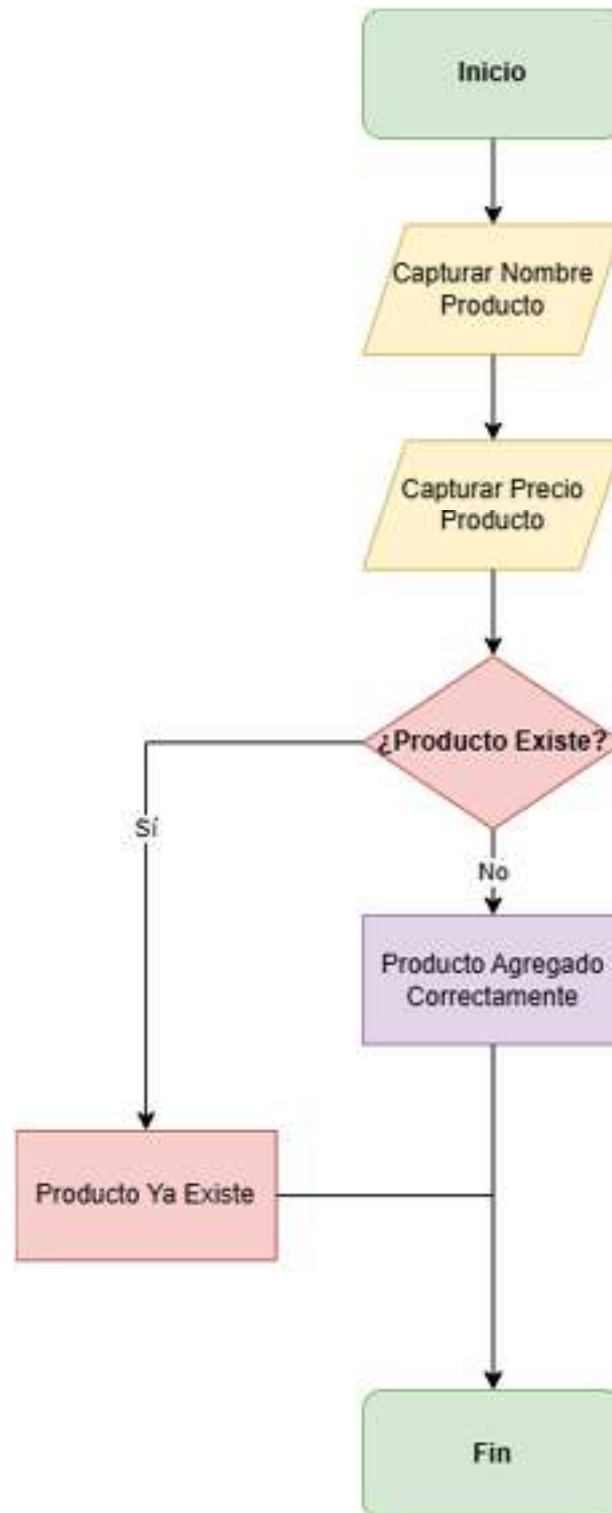
- Login



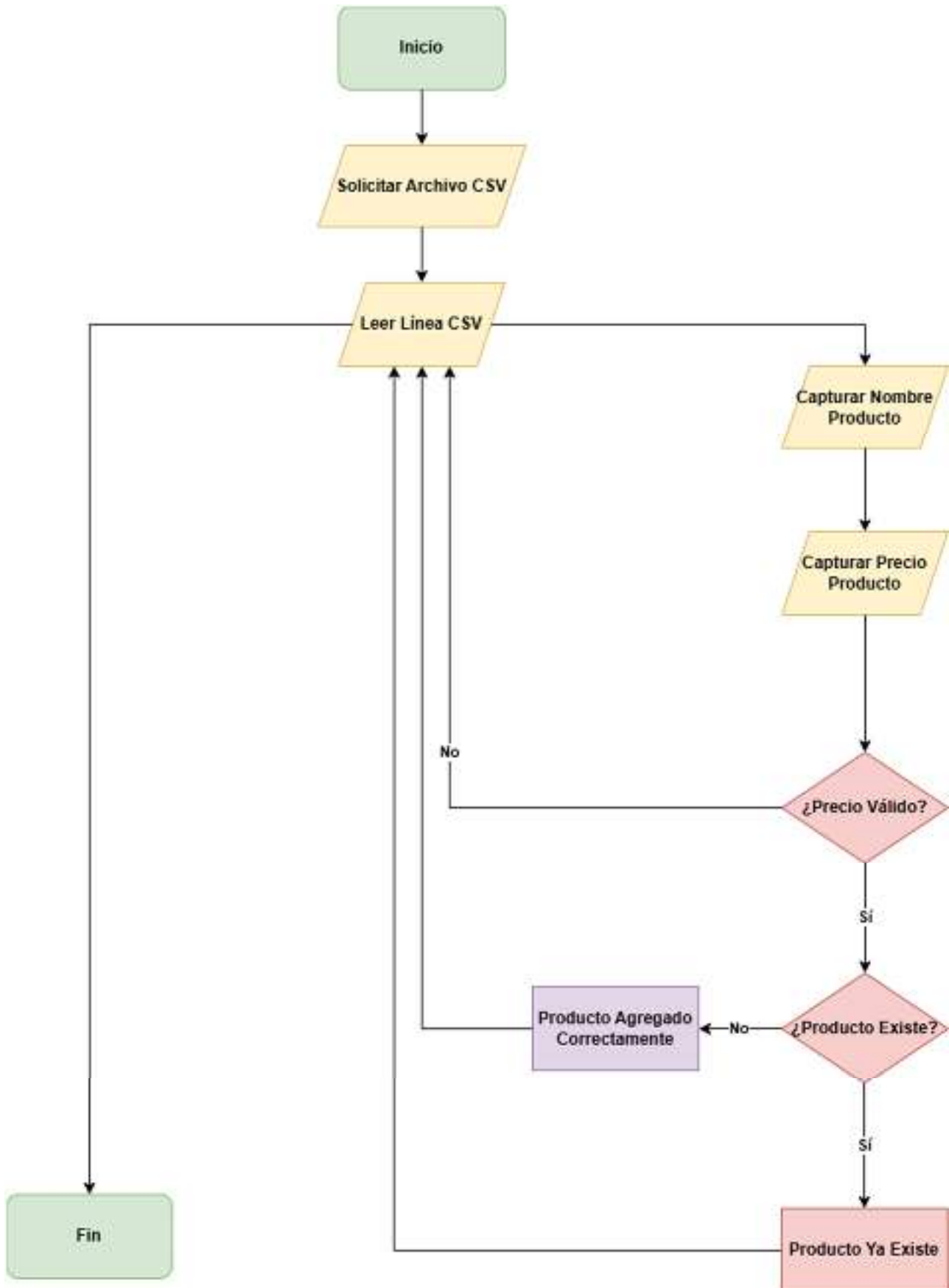
- Menú principal



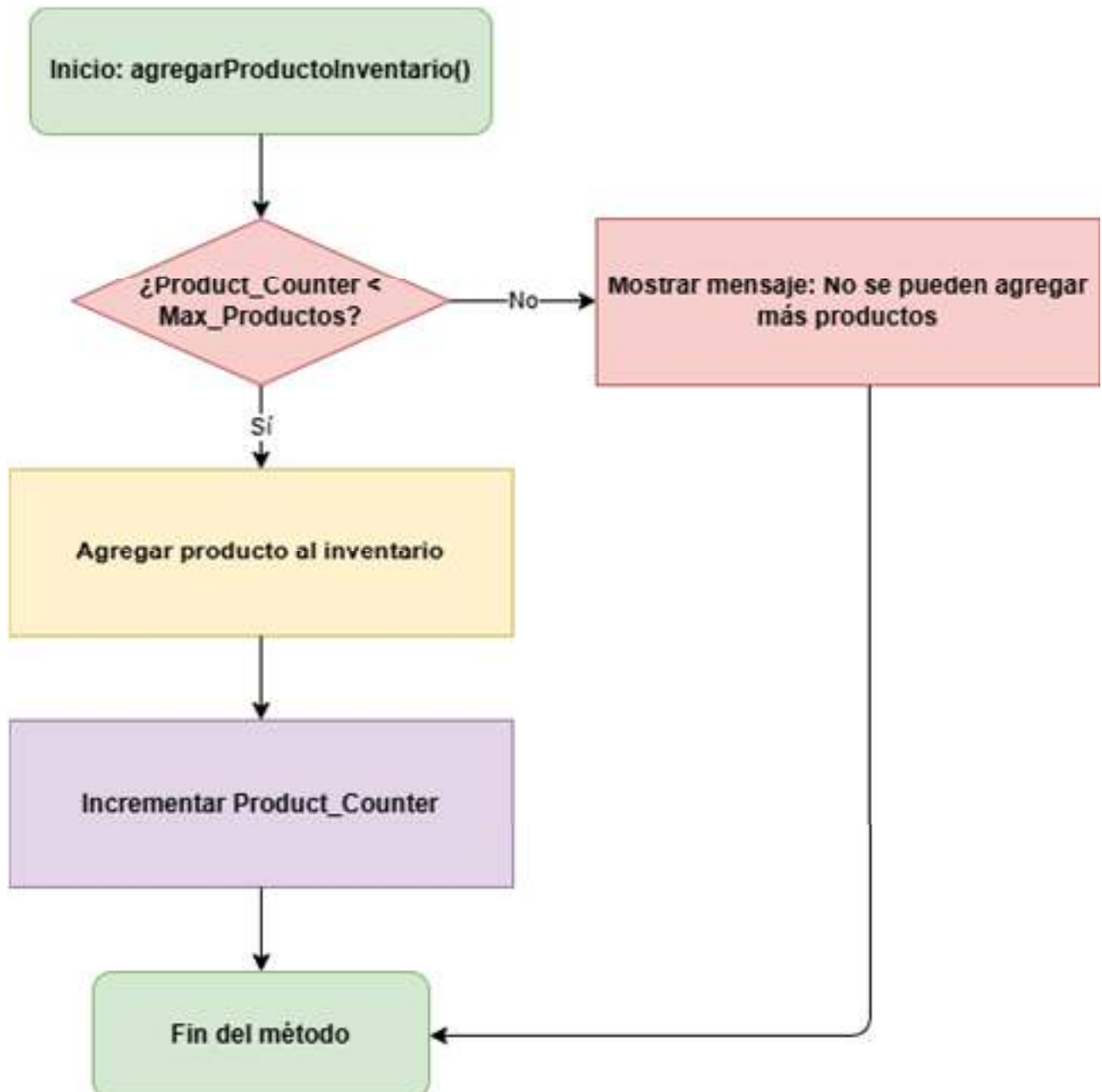
- Agregar producto



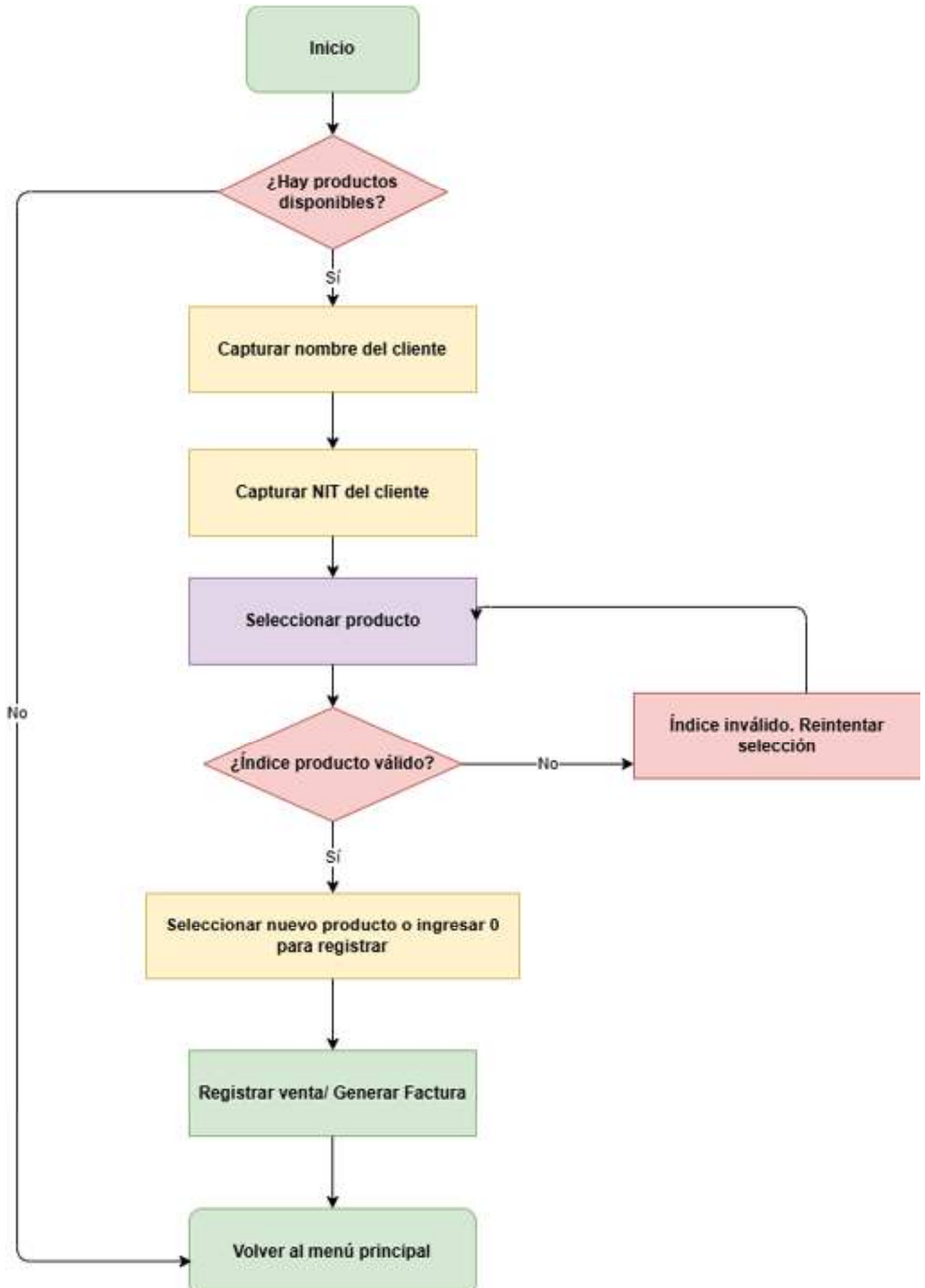
- Agregar producto (masiva)



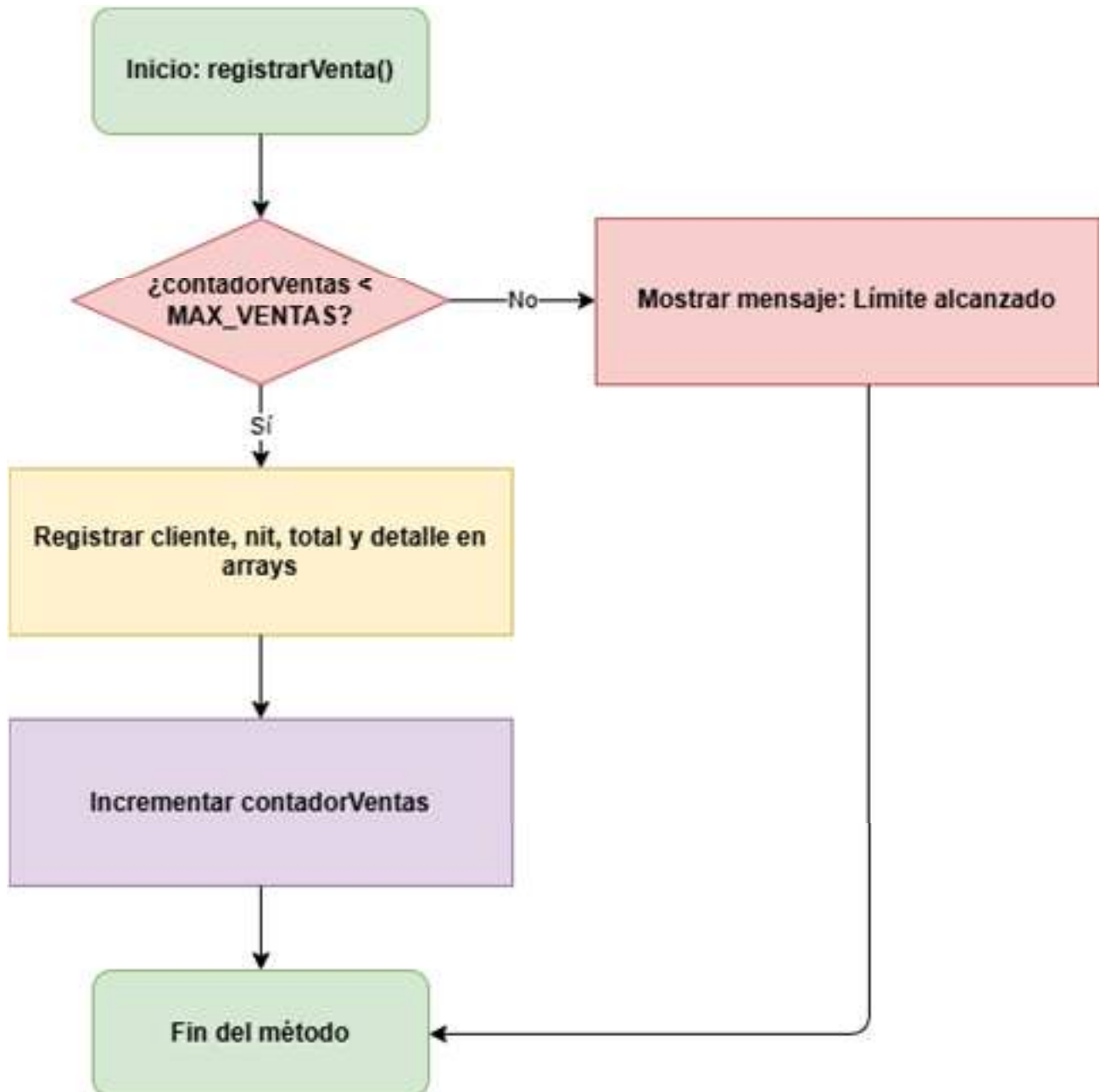
- **Agregar Producto a Inventario**



- Realizar venta

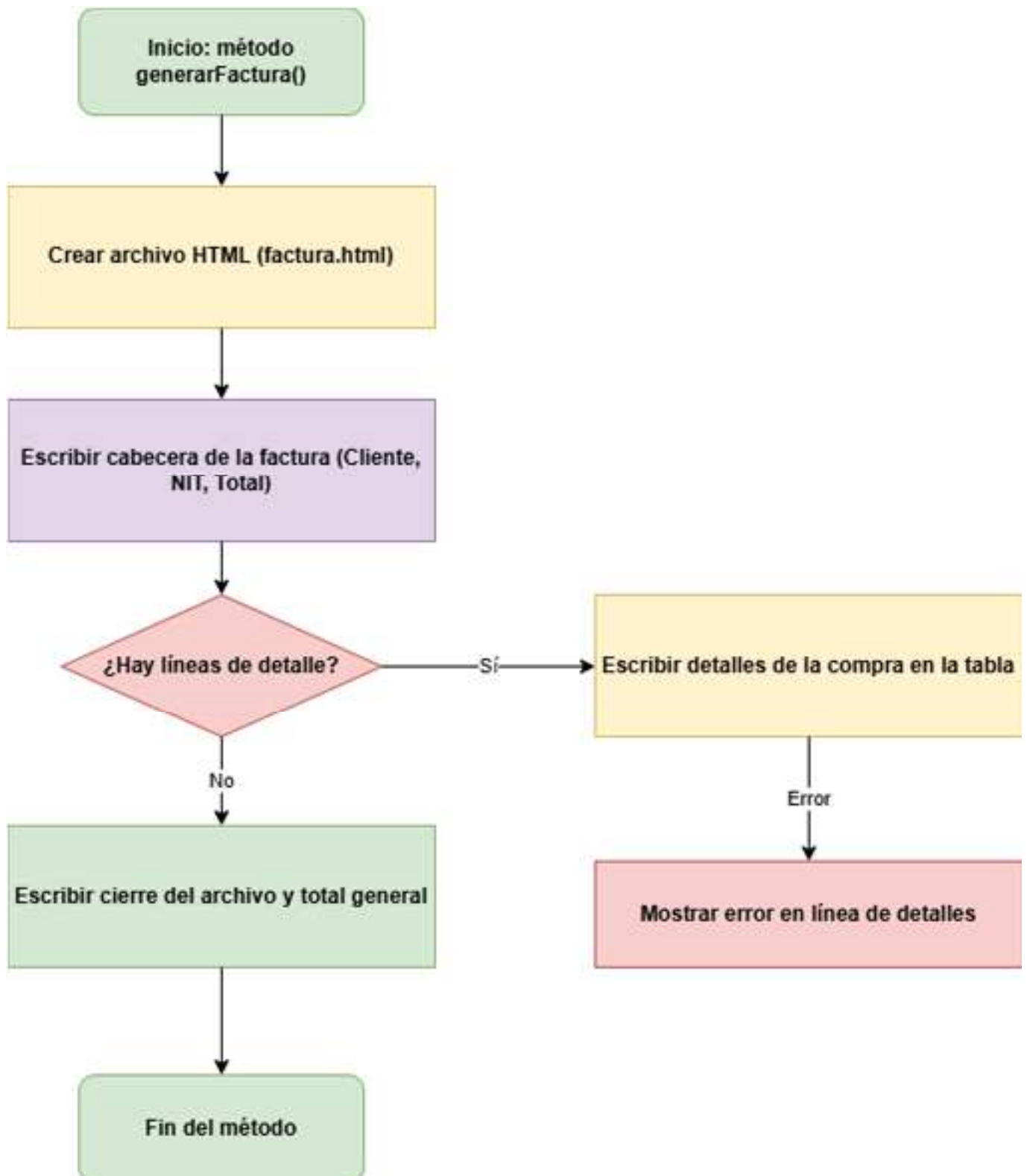


- Registrar Venta

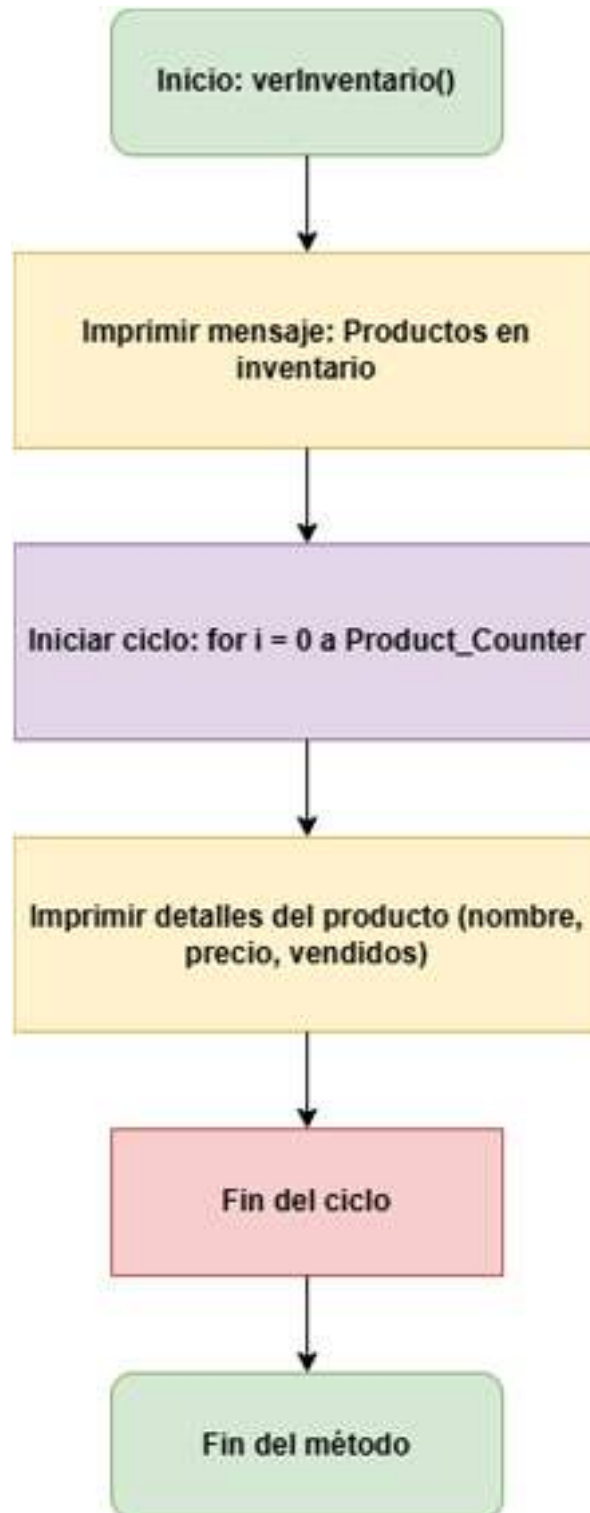




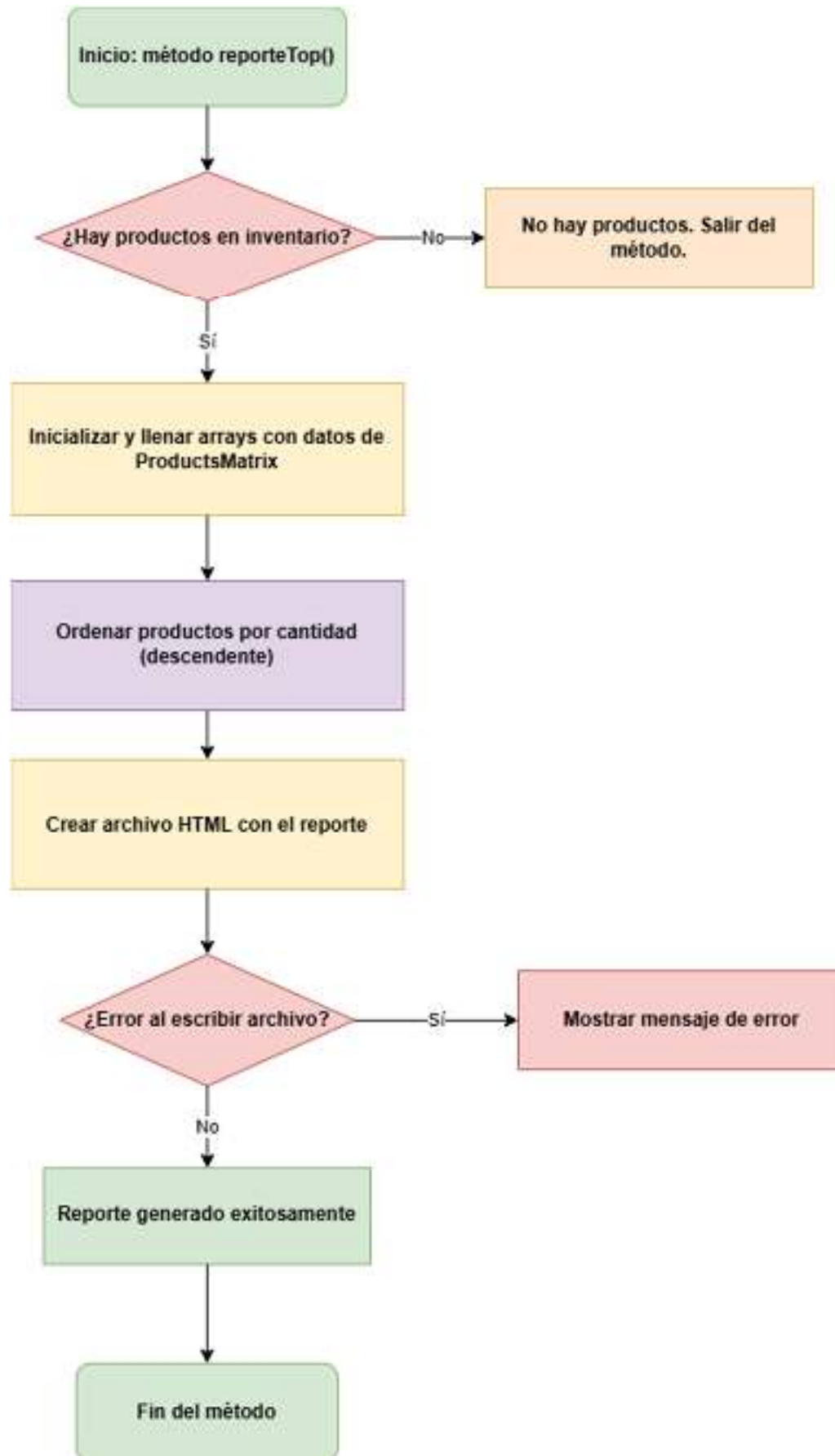
- Generar Factura



- Ver Inventario



- Reporte Top 5



- **Reporte Histórico**

