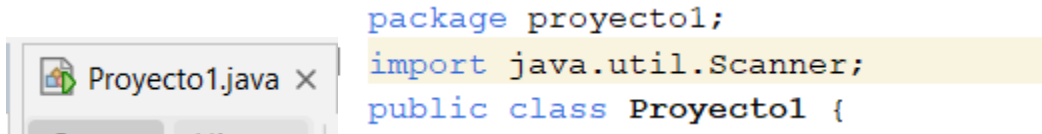


# Manual Técnico

## 1. Estructura del Proyecto:

- Archivo Principal: Contiene el menú, lógica general y operaciones del usuario. Llamada Proyecto1.



- En la primera sección se pueden observar un import java.util.Scanner que sirve para importar Scanner esto lee entradas del usuario desde la consola.

Y los static son variables globales en donde se declaran arreglos de objetos con capacidad de 100 máxima de productos.

```
package proyecto1;
import java.util.Scanner;
public class Proyecto1 {
    static Producto[] productos = new Producto[100];
    static int totalProductos = 0;
```

- También aparece un static string usuario donde se colocó un nombre, en este caso fue míó, y este sirve cuando se muestra la bitácora de las acciones realizadas para que se vea quien hizo los cambios.

```
    static int totalProductos = 0;
    static String usuario = "Alejandro";
    static String[] bitacora = new String[1
```

- Menú Principal: Se utilizo un bucle do-while que muestra el menú de opciones y permite repetir hasta que el usuario seleccione salir (opción 8).

Además, se usa un switch para controlar las diferentes funcionalidades según la opción seleccionada.

```

do{
    //Menu Principal
    System.out.println("-----Menu Principal-----");
    System.out.println("1. Agregar producto");
    System.out.println("2. Buscar producto");
    System.out.println("3. Eliminar producto");
    System.out.println("4. Registrar venta");
    System.out.println("5. Bitacora");
    System.out.println("6. Generacion de reportes");
    System.out.println("7. Datos del estudiante");
    System.out.println("8. Salir");

    try{
        System.out.print("Selecciona una opcion: ");
        opcion = Integer.parseInt(scanner.nextLine());
    }catch (NumberFormatException e) {
        System.out.println("Entrada invalida. Por favor, ingresa un numero del 1 al 8.");
        opcion = 0;
    }

    switch (opcion){

```

- Case 1: Este agrega productos, preguntando cierta información antes de agregarla y también tiene la función para reconocer cuando se coloca algo que no es permitido o cuando no se ingresa ningún valor.

```

case 1:{
    if (totalProductos >= productos.length) {
        System.out.println("No se pueden agregar mas productos.\n");
        break;
    }

    System.out.println("----- Agregar Producto -----");
    System.out.print("Codigo del producto: ");
    String codigo = scanner.nextLine();

    boolean existe = false;
    for (int i = 0; i < totalProductos; i++) {
        if (productos[i].getCodigo().equalsIgnoreCase(codigo)) {
            existe = true;
            break;
        }
    }

    if (existe) {
        System.out.println("El codigo ya existe.\n");
        registrarBitacora("Agregar producto", false);

        break;
    }

    System.out.print("Nombre del producto: ");
    String nombre = scanner.nextLine();

    System.out.print("Categoria: ");
    String categoria = scanner.nextLine();

```

- Case 2: Acá se busca el producto mediante su categoría, nombre o código.

```
case 2:
    System.out.println("-----Buscar producto-----");
    System.out.println("Ingresa nombre, categoria o codigo del producto: ");
    String criterio = scanner.nextLine().toLowerCase();

    if (criterio.isEmpty()) {
        System.out.println("Debe ingresar un criterio para buscar.");
        break;
    }

    boolean encontrado = false;
    for (int i=0; i < totalProductos; i++){
        Producto p = productos[i];
        if (p.getNombre().toLowerCase().contains(criterio) ||
            p.getCodigo().toLowerCase().contains(criterio) ||
            p.getCategoria().toLowerCase().contains(criterio)) {
            System.out.println(p);
            encontrado = true;
        }
    }
```

Además, mira si el código o nombre esta registrado o no mediante una condicional.

```
if(!encontrado){
    System.out.println("Producto no encontrado");
    registrarBitacora("Buscar producto", encontrado);
}
break;
```

- Case 3: Acá se eliminan los productos

```
case 3:
    System.out.println("-----Eliminar producto-----");
    System.out.println("Ingrese el codigo del producto a eliminar: ");
    String codigoeliminar = scanner.nextLine().trim();

    if (codigoeliminar.isEmpty()) {
        System.out.println("El codigo no puede estar vacio.");
        break;
    }
```

- Case 4: Registra las ventas que se van desarrollando en el transcurso del que se ejecuta el programa.

```

case 4:
    System.out.println("-----Registro de la venta-----");
    System.out.println("Ingrese el codigo del producto: ");
    String codigoVenta = scanner.nextLine().trim();

    Producto productoVenta = null;
    for(int i=0; i < totalProductos; i++){
        if (productos[i].getCodigo().equalsIgnoreCase(codigoVenta)){
            productoVenta = productos[i];
            break;
        }
    }
    if (productoVenta == null){
        System.out.println("Producto no encontrado");
        registrarBitacora("Registrar venta", false);
        break;
    }

    System.out.println("Producto encontrado: " + productoVenta);

    int cantidadVenta = 0;
    boolean cantidadValida = false;

```

- Case 5: Acá se mostrarán las acciones generales que se hacen en el programa, aunque sean errores los que se generen también los guardará y los mostrará cuando se solicite, además muestra el nombre que en un inicio se asignó para que muestre quien lo hizo.

```

case 5:
    System.out.println("-----Bitacora-----");
    if (totalRegistrosBitacora == 0) {
        System.out.println("No se han registrado acciones aun.");
    } else {
        for (int i = 0; i < totalRegistrosBitacora; i++) {
            System.out.println(bitacora[i]);
        }
    }
    break;

```

- Case 6: Acá se generan los reportes en PDF de las ventas o del stock

```

case 6:
    System.out.println("-----Generar reportes-----");
    System.out.println("1. Reporte de stock");
    System.out.println("2. Reporte de ventas");
    System.out.println("Seleccione una opcion: ");

    String input = scanner.nextLine().trim();
    int opcionReporte = 0;

    try {
        opcionReporte = Integer.parseInt(input);
    } catch (NumberFormatException e) {
        System.out.println("Entrada invalida. Por favor, ingresa un número (1 o 2).");
        break;
    }

```

- Case 7: Acá se muestra la información del estudiante que en este caso es mía.

case 7:

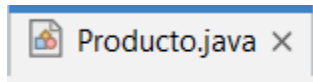
```
System.out.println("-----Datos del Estudiante-----");
System.out.println("Alejandro Emmanuel Alvarez Velasquez");
System.out.println("Carnet: 202404224");
System.out.println("Laboratorio IPC - Seccion: A");
System.out.println("Github: alejandroalvarezv");
break;
```

- Case 8: Cuando el usuario coloca el numero 8 en el menú automáticamente termina el programa mediante while y de ultimo hay un scanner.close que sirve para liberar memoria y no cargar el sistema.

```
case 8:
    System.out.println("Salir");
    break;

}while (opcion !=8);
scanner.close();
```

- Clase 1: Llamada Producto, en ella están todas las acciones con los productos con atributos y métodos que se desarrollaron en todo el proyecto. Dentro de esta clase se llama a proyecto1 para hacer la conexión entre clases.



```
package proyecto1;
public class Producto {
```

- Atributos: La clase Producto representa un modelo de datos para cada producto que se gestiona en el inventario. Contiene la información básica de un producto y algunos métodos para acceder y modificar estos datos.

Estos son los atributos que definen a un producto:

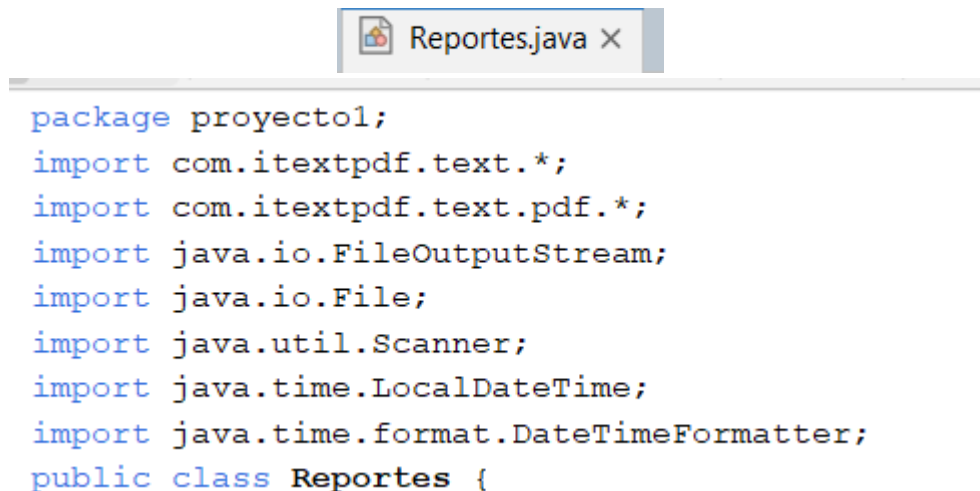
```
private String nombre;  
private String categoria;  
private double precio;  
private int cantidad;  
private String codigo;
```

Acá se encapsulan

- Constructor: Es el constructor de la clase. Se llama cuando se crea un nuevo producto. Además, recibe los valores de todos los atributos e inicializa el objeto con ellos.

```
public Producto(String nombre, String categoria, double precio, int cantidad, String codigo) {
```

- Clase 2: Llamada Reportes, esta es una clase que genera reportes PDF de stock y ventas utilizando la librería iText, y esta de igual manera que la anterior llama a proyecto1 para hacer la conexión, aunque por defecto se realiza.



```
package proyecto1;  
import com.itextpdf.text.*;  
import com.itextpdf.text.pdf.*;  
import java.io.FileOutputStream;  
import java.io.File;  
import java.util.Scanner;  
import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
public class Reportes {
```

- Acá se muestra lo demás del código en la clase llamada Reportes.java

```

public static void generarReporteStockPDF(Producto[] productos, int totalProductos){
    try{
        String fechaHora = LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd_MM_yyyy_HH_mm_ss"));
        String nombreArchivo = fechaHora + "_Stock.pdf";

        Document document = new Document();
        PdfWriter.getInstance(document, new FileOutputStream(nombreArchivo));
        document.open();

        document.add(new Paragraph("Reporte de Stock", FontFactory.getFont(FontFactory.HELVETICA_BOLD, 18)));
        document.add(new Paragraph("Fecha de generación: " + fechaHora.replace("_", ":"), FontFactory.getFont(FontFactory.HELVETICA, 12)));
        document.add(new Paragraph("\n"));

        PdfPTable tabla = new PdfPTable(5);
        tabla.setWidthPercentage(100);
        tabla.addCell("Codigo");
        tabla.addCell("Nombre");
        tabla.addCell("Categoria");
        tabla.addCell("Precio (Q)");
        tabla.addCell("Cantidad");
    }
}

```

## 2. Softwares necesarios:

- ✓ Sistema Operativo:
  - Windows 10/11
- ✓ Java Development Kit (JDK):
  - Version 8 o superior
- ✓ IDE:
  - Netbeans 12 en adelante
- ✓ Librerías externas:
  - ITextPDF para poder generar reportes en formato PDF.

## 3. Hardware necesarios:

- ✓ Procesador: Intel 5 en adelante.
- ✓ Memoria RAM: 4GB.
- ✓ Almacenamiento 500 MB en adelante.
- ✓ Una computadora.
- ✓ Un mouse si en caso se esta utilizando computadora de escritorio.
- ✓ Un teclado si en caso se esta utilizando computadora de escritorio.

#### 4. Métodos más importantes:

- registrarBitacora(String tipoAccion, boolean fueCorrecta)

Ubicación: Clase Proyecto1

Registra una acción realizada por el usuario en un arreglo llamado bitacora[], indicando si fue correcta o errónea, junto con la fecha, hora y nombre del usuario.

Parámetros:

tipoAccion: nombre de la acción (por ejemplo: "Agregar producto")

fueCorrecta: indica si la acción fue exitosa (true) o no (false)

Permite llevar un registro interno de todo lo que hace el usuario, útil para auditoría o depuración.

- generarReporteStockPDF(Producto[] productos, int totalProductos)

Ubicación: Clase Reportes

Genera un archivo PDF con una tabla que contiene todos los productos en stock, incluyendo su nombre, código, categoría, precio y cantidad disponible.

Parámetros:

productos[]: arreglo que contiene todos los productos

totalProductos: número total de productos actuales

Permite al usuario generar un reporte visual de todo el inventario actualizado.



- `generarReporteVentasPDF()`

Ubicación: Clase Reportes

Lee el archivo `ventas.txt` y genera un PDF con una tabla que muestra todas las ventas realizadas, incluyendo código del producto, cantidad, fecha y total vendido.

Permite consultar todas las ventas realizadas de forma organizada y legible.

- `disminuirCantidad(int cantidadVendida)`

Ubicación: Clase Producto

Resta la cantidad vendida del stock disponible del producto.

Parámetros:

`cantidadVendida`: número de unidades que se vendieron

Mantiene actualizado el stock real del producto después de cada venta.

- `getCantidad()`, `getPrecio()`, `getCodigo()`, `getNombre()`, `getCategoria()`

Ubicación: Clase Producto

Son métodos de acceso (getters) que devuelven los atributos privados del producto.

Permiten acceder a los datos del producto desde otras clases sin violar el encapsulamiento.