Manual Técnico

El programa consiste en la simulación de una tienda en la cual se pueden realizar varias acciones mediante el menú de opciones. Dicho menú está creado para que el usuario pueda personalizar su compra y guardarla. También permite generar reportes de ventas y reportes de inventario mediante opciones dadas en dicho menú, los reportes de inventario se darán mediante un PDF el cual se creará cuando el usuario lo pida.

Para iniciar y compilar el código necesario saber que necesitamos tener previamente instalada la aplicación de NetBeans pues en dicha aplicación está instalado el compilador que nos ayudará a leer el lenguaje de java. (Se adjunta foto del logo de NetBeans).

1. **<u>Librerías.</u>**

Una vez estemos dentro de nuestro programa NetBeans necesitamos importar las librerías las cuales van a ayudarnos con el desarrollo del código, las que se muestran a continuación de la línea 3 a la línea 18 son las librerías utilizadas en desarrollo, cada una con una función distinta.

- **Document:** Nos ayuda a la creación y al manejo del documento PDF que necesitamos crear.
- **DocumentException:** Se encarga en su mayoría del control de errores al trabajar con archivos PDF.
- **PdfWriter:** Es el que se encarga de escribir lo que proponemos en el archivo PDF.
- **Paragraph:** Es el que inserta los párrafos.
- **Element:** Controla e importa los elementos como el texto o imágenes.
- **FontFactory:** Crea y gestiona fuentes.
- **PageSize:** Se encarga de definir el tamaño de página.
- **PDFTable:** Crea e inserta tablas dentro del PDF.
- **PdfCell:** Define las celdas dentro de las tablas que crea el comando anterior.
- **Java.io*:** Maneja el flujo de archivos ya sea de entrada o salida.
- **LocalDataTime:** Esta librería nos proporciona la hora para una salida de consola.
- **DataTimeFormatter:** Nos proporciona el formato de las fechas y horas.
- **Scanner:** Es el que permite la interacción del usuario con la consola.

```
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.PdfWriter;
import com.itextpdf.text.Element;
import com.itextpdf.text.Font;
import com.itextpdf.text.FontFactory;
import com.itextpdf.text.PageSize;
import com.itextpdf.text.Phrase;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfPCell;

import java.io.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;
```

2. Creación de variables publicas

Nuestro código muestra la creación de las variables publicas utilizadas desde la línea 20 hasta la línea 31 para todo el código, las cuales serán utilizadas en el proceso del código para emplear métodos, asignaciones, validaciones y ejecuciones de variables.
Se puede apreciar que también existe un public static para un scanner pues este será utilizado en gran parte del código para que el usuario pueda interactuar con la consola.

```
public class Proyecto1 {

    static int elementosMax = 100;
    static int bitacoraMax = 200;
    static String codigo [] = new String [elementosMax];
    static String nombres [] = new String [elementosMax];
    static String categorias [] = new String[elementosMax];
    static float precio [] = new float [elementosMax];
    static int stocks [] = new int [elementosMax];
    static String  accionesBitacora [] = new String[bitacoraMax];
    static int totalProductos = 0;
    static int totalBitacora = 0;

    static Scanner entrada = new Scanner(System.in);
```

3. Creación del Menú y opciones para el menú.

Comenzamos creando un variable tipo int para inicializar nuestras opciones en 0, seguido de un cilo do-while el cual nos ayudará a recorrer todo el menú y sus múltiples opciones.

```
int opcion =0;

do{
    System.out.println("\n=== Bienvenido al menu de la tienda ===");
    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. Generar reportes");
    System.out.println("6. Ver datos del estudiante");
    System.out.println("7. Bitacora");
    System.out.println("0. Salir");
    System.out.print("Seleccione una opcion: ");
```

Después de haber enviado las opciones a consola para que el usuario pueda elegir una opción procedemos a hacer un switch para que el menú pueda ser interactivo asignándole una ejecución a cada una de las opciones de nuestro menú, dicho switch va a estar contenido dentro de un try-catch en el cual vamos a darle la especificación de que únicamente admita valores numéricos, esto con el fin de que el usuario digite exclusivamente valores numéricos al momento usar el menú. El código seguirá repitiéndose y pidiendo opciones hasta que el usuario decida salir y digite la opción asignada.

```
        try{
            opcion = Integer.parseInt(entrada.nextLine());
            switch(opcion){
                case 1:
                    agregarProducto();
                    break;
                case 2:
                    buscarProducto();
                    break;
                case 3:
                    eliminarProducto();
                    break;
                case 4:
                    registrarVenta();
                    break;
                case 5:
                    generarReporte();
                    break;
                case 6:
                    verDatosEstudiante();
                    break;
                case 7:
                    bitacora();
                    break;
                case 8:
                    System.out.println("Saliste del menu principal");
                    break;
                default :
                    System.out.println("OPCION INVALIDA, DIGITE UN NUMERO DEL MENU");
            }
        }catch(NumberFormatException e){
            System.out.println("Error: Debe ingresar un numero valido");
        }
    }while(opcion != 8);
}
```

4. **<u>Creación de clases publicas y privadas:</u>**

   Para cada una de las opciones de nuestro menú se necesita crear clases de manera independiente pues cada una de estas va a tener una función distinta a las otras, entonces dependiendo de lo que el código requiera cada una de las opciones tendrá su propia clase publica o/y privada.
   Las clases creadas fueron las siguientes:

```
public static void agregarProducto(){
public static void buscarProducto(){
public static void eliminarProducto(){
public static void generarReporte(){
```

```
private static void generarReporteStockPDF(
private static void generarReporteVentasPDF(
public static void verDatosEstudiante()
public static void registrarAccion(
public static void bitacora()
private static void addHeader(
private static String safe(
```

5. Desarrollo del código:

5.1. Agregar producto

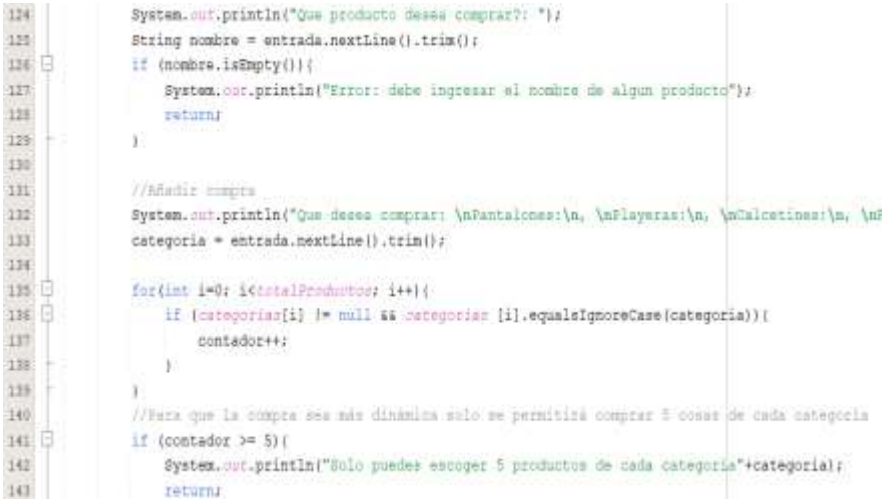
El desarrollo empieza desde la línea 94 en el cual antes de agregar un producto se verifica que el usuario esté dentro del rango de productos, de lo contrario no se podrán hacer más transacciones.
Si dichos parámetros se cumplen, procedemos con las variables que serán utilizadas para el desarrollo y validación de la compra.

```
94      if (totalProductos >= elementosMax){
95          System.out.println("Llego al limite de compra, por favor haga una nueva compra");
96          return;
97      }
98
99      System.out.println("Agregue un producto a su compra :)");
100
101
102     boolean codigoUnico = false;
103     int contador = 0, stockNuevo;
104     String nuevoCodigo, categoria;
105     float precioNuevo;
```

A partir de la línea 107 usaremos el ciclo do-while para hacer que el código de producto creado sea único, esto nos ayuda a mantener el control de inventario.

```
107     do{
108     System.out.println("Ingrese el codigo del articulo: ");
109     nuevoCodigo = entrada.nextLine().trim();
110     if (nuevoCodigo.isEmpty()){
111         System.out.println("Error: el codigo esta vacio, por favor digite un codigo valido");
112     }
113
114     codigoUnico = true;
115     for(int i = 0; i <totalProductos; i++){
116         if(codigo[i] != null && codigo[i].equalsIgnoreCase(nuevoCodigo))
117             System.out.println("Error: este codigo ya existe");
118         codigoUnico = false;
119         break;
120     }
121     }while(!codigoUnico);
```

Ya validamos que el código de nuestro producto sea único, ahora lo que hacemos es agregar un producto a nuestra lista y para que el menú sea más dinámico hacemos que únicamente se puedan comprar 5 artículos por cada categoría, si la compra excede los 5 artículos el programa nos dará un error.



```
124         System.out.println("Que producto desea comprar?: ");
125         String nombre = entrada.nextLine().trim();
126         if (nombre.isEmpty()){
127             System.out.println("Error: debe ingresar el nombre de algun producto");
128             return;
129         }
130
131         //Añadir compra
132         System.out.println("Que desea comprar: \nPantalones:\n, \nPlayeras:\n, \nCalcetines:\n, \n
133         categoria = entrada.nextLine().trim();
134
135         for(int i=0; i<totalProductos; i++){
136             if (categorias[i] != null && categorias [i].equalsIgnoreCase(categoria)){
137                 contador++;
138             }
139         }
140         //Para que la compra sea más dinámica solo se permitirá comprar 5 cosas de cada categoria
141         if (contador >= 5){
142             System.out.println("Solo puedes escoger 5 productos de cada categoria"+categoria);
143             return;
```

Tenemos las validaciones y el producto que deseamos comprar, ahora necesitamos asignarle un precio y guardar la compra dentro de nuestro inventario para hacer la cuenta de los productos, para los precios tendremos que usar nuevamente el arreglo try-catch para que el precio sea de manera numérica y no textual.

```
146         System.out.println("Precio: ");
147         String precios = entrada.nextLine().trim();
148         try{
149             precioNuevo = Float.parseFloat(precios);
150             if(precioNuevo < 0){
151                 System.out.println("El precio no puede ser negativo");
152                 return;
153             }
154         }catch (NumberFormatException e){
155             System.out.println("Error: digita un precio valido");
156             return;
157         }
158
159         System.out.print("Stock: ");
160         String stockStr = entrada.nextLine().trim();
161          try{
162         stockNuevo = Integer.parseInt(stockStr);
163         if (stockNuevo < 0){
164             System.out.println("El stock tiene que ser un valor positivo, corrijalo por favor.");
165             return;
166         }
167         }catch (NumberFormatException e){
168         System.out.println("Stock invalido.");
169         return;
170
171         }
172         codigo[totalProductos] = nuevoCodigo;
173         nombres[totalProductos] = nombre;
174         categorias[totalProductos] = categoria;
175         precio[totalProductos] = precioNuevo;
176         stocks[totalProductos] = stockNuevo;
177         totalProductos++;
178
179         System.out.println("Listo! su producto fue agregado a la compra");
180
```

5.2. Buscar producto

Para buscar un producto empezamos el desarrollo desde la línea 188 y para realizar dicha búsqueda necesitamos recorrer el vector con un ciclo for el cual contendrá una clase booleana que tendrá la función de identificar si el producto fue encontrado o no, si el producto fue encontrado nos saltará un mensaje en consola el cual mostrará cual fue el producto y sus características.

```
188             System.out.println("¿Que desea buscar?: ");
189             String dato = entrada.nextLine().trim();
190
191             boolean encontrado = false;
192             int indice = -1;
193
194             for(int i=0; i<totalProductos; i++){
195                 if(categorias[i] != null && categorias[i].equalsIgnoreCase(dato)){
196                     encontrado = true;
197                     indice = i;
198                     break;
199                 }
200             }
201
202             if(!encontrado){
203                 System.out.println("No se encontro su producto, intente de nuevo.");
204             } else{
205                 System.out.println("Genial! su articulo fue encontrado:");
206                 System.out.println("1. Codigo: "+codigo[indice]);
207                 System.out.println("2. Codigo: "+nombres[indice]);
208                 System.out.println("3. Codigo: "+categorias[indice]);
209                 System.out.println("4. Codigo: "+precio[indice]);
210                 System.out.println("5. Codigo: "+stocks[indice]);
211             }
212         }
213
```

5.3. Eliminar producto

Para eliminar un producto, al igual que para buscar necesitamos recorrer todo nuestro vector e identificar cual es el producto a eliminar dentro del índice, una vez identificado se correrá el vector una fila y se reducirá el índice vaciando el espacio del vector.

5.4. Registrar venta

Es un proceso simple, pues para registrar una venta necesitamos guardar dentro de un vector el producto deseado por el usuario, el código y el precio.

Si por alguna razón el precio es una cantidad menor o igual a 0 la venta no se registrará porque no es posible registrar una venta negativa. Si el usuario digita todo correctamente se hará una pequeña operación matemática la cual determinará el total de la venta realizada en esa transacción.

```
        System.out.println("Ingrese el código del producto para registrar la venta: ");
        String codProducto = entrada.nextLine().trim();
        int posicion = -1;
        for(int i=0; i<totalProductos; i++){
            if(codigo[i].equalsIgnoreCase(codProducto)){
                posicion = i;
                break;
            }
        }
        if(posicion == -1){
            System.out.println("No existe ningun producto con este codigo");
            return;
        }

        boolean validarCantidad = false;
        do{
            System.out.println("Ingrese la cantidad vendida: ");
            int cantVenta = Integer.parseInt(entrada.nextLine());
            if(cantVenta <= 0){
            System.out.println("Necesita ingresar un valor positivo");
        }else if(cantVenta > totalProductos){
                System.out.println("La venta supera a la cantidad de los productos disponibles");

                totalProductos -= cantVenta;
                int totalVenta = (int)(cantVenta*precio[posicion]);

                System.out.println("La venta total es de :"+totalVenta);
                System.out.println("Venta Registrada de manera exitosa");
                System.out.println("Producto: "+nombres[posicion]);
                System.out.println("Total vendido: "+totalVenta);
                System.out.println("Stocks: "+stocks[posicion]);
                validarCantidad = true;
            }

        }while(!validarCantidad);
```

5.5. Generar Reporte

Para generar el reporte de inventario es necesario crear 2 clases privadas las cuales harán la función pueda ejecutarse

```
        try{
            String fecha = LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd_MM_yyyy_HH_mm_ss"));
            String stockArchivo = fecha +"_Stock.pdf";
            String ventaArchivo = fecha +"_Venta.pdf";

            generarReporteStockPDF(stockArchivo);
            generarReporteVentasPDF(ventaArchivo);

            System.out.println("Reportes: ");
            System.out.println(" " +stockArchivo);
            System.out.println(" " +ventaArchivo);

        }catch(Exception e){
            System.out.println("Error al generar el reporte"+e.getMessage());
        }

    }
```

Dentro de las clases privadas realizaremos el proceso de la creación de los PDF citando las librerías instaladas previamente, las cuales también ya fueron mencionadas con anterioridad, dentro de este bloque de código editaremos y le daremos parámetros a nuestro archivo PDF con características básicas como el tamaño de página, letra y aparte también será necesario insertar una tabla en la cual se muestra el reporte del inventario de stocks.

```
        Document documento = new Document(PageSize.LETTER, 36,36,36,36);
        PdfWriter.getInstance(documento, new java.io.FileOutputStream(nomArchivo));

        documento.open();

        Paragraph titulo = new Paragraph("Estos son los reportes generados para stock", FontFactory.getFont(FontFactory.HELVETICA_BO
        documento.add(titulo);
        documento.add(new Paragraph(" "));

        PdfPTable tablapdf = new PdfPTable(5);
        tablapdf.setWidthPercentage(100);
        tablapdf.setWidths(new float []{25f,20f,22f,15f,15f,});

        addHeader(tablapdf, "Nombre :");
        addHeader(tablapdf, "Codigo :");
        addHeader(tablapdf, "Categoria :");
        addHeader(tablapdf, "Precio : ");
        addHeader(tablapdf, "Stock :");

        Font fFila = FontFactory.getFont(FontFactory.COURIER, 10);
        for (int i = 0; i<totalProductos; i++){

            tablapdf.addCell(new Phrase(safe(nombres[i]), fFila));
            tablapdf.addCell(new Phrase(safe(codigo[i]), fFila));
            tablapdf.addCell(new Phrase(safe(categorias[i]), fFila));
            tablapdf.addCell(new Phrase(String.format("%.2f", precio [i]), fFila));
            tablapdf.addCell(new Phrase(String.valueOf(stocks[i]), fFila));
        }

        documento.add(tablapdf);
        documento.close();
    }
```

```
        Document documento = new Document(PageSize.LETTER, 36,36,36,36);
        PdfWriter.getInstance(documento, new java.io.FileOutputStream(nomArchivo));
        documento.open();

        Paragraph tituloDoc = new Paragraph(" Reporte de ventas ", FontFactory.getFont(FontFactory.HELVETICA_BOLD,16));
        tituloDoc.setAlignment(Element.ALIGN_LEFT);
        documento.add(tituloDoc);
        documento.add(new Paragraph(" "));

        PdfPTable tabla2 = new PdfPTable(5);
        tabla2.setWidthPercentage(100);
        tabla2.setWidths(new float[]{20f,20f,10f,10f,24f});

        addHeader(tabla2, " Nombre: ");
        addHeader(tabla2, " Codigo: ");
        addHeader(tabla2, " Cantidad total: ");
        addHeader(tabla2, " Total (en quetzales): ");
        addHeader(tabla2, " Fecha y hora: ");

        Font fFila = FontFactory.getFont(FontFactory.COURIER, 10);
        File f = new File("Reporte de ventas.txt");
        if (!f.exists()){
            documento.add(new Paragraph("ERROR: NO SE REGISTRO NINGUNA VENTA",FontFactory.getFont(FontFactory.HELVETICA_OBLIQUE, 11
            return;
        }
        float totalGeneral = 0f;

        try(BufferedReader lectura = new BufferedReader(new FileReader(f))){
            String linea;
            while((linea =lectura.readLine()) !=null){
                String[] p = linea.split(" ");
                if (p.length < 6) continue;

                String codigoProd = p[0];
                String nomProd = p[1];
                String cantProd = p[2];
```

También añadimos clases las cuales denominamos como auxiliares las cuales se encargan de editar y almacenar los cambios que se realicen dentro del archivo PDF.

```
424     private static void addHeader(PdfPTable tabla, String txt) {
425         Font f = FontFactory.getFont(FontFactory.HELVETICA_BOLD, 11);
426         PdfPCell sc = new PdfPCell(new Phrase(txt, f));
427         sc.setHorizontalAlignment(Element.ALIGN_CENTER);
428         tabla.addCell(sc);
429     }
430
431     private static String safe(String datosfinales) { return (datosfinales == null) ? "" : datosfinales; }
432
433 }
```

5.6. Ver datos del estudiante

En esta sección lo único que nos pide es colocar nuestros datos, entonces solamente imprimimos los datos personales y los del curso.

```
395             System.out.println("Nombre: Juan Jose María González Tuch");
396             System.out.println("Carnet: 202300700");
397             System.out.println("Proyecto 1 IPC1");
398         }
```

5.7. Bitácora

Lo que la bitácora solicita es un historial de las acciones que realice el usuario en el transcurso de la interacción con el código, entonces lo que necesitamos hacer es llamar a nuestras clases públicas y asignarle un parámetro a cada.

También usamos nuestra librería LocalDataTime para registrar la hora y fecha en la que se realizan las acciones. Teniendo todas las acciones guardadas se ejecutan en una variable la cual denominamos inicio y finalizamos agregando un elemento a nuestra bitácora usando el comando ++;

Asignamos una condición la cual dice que, si en la bitácora no hay ningún elemento ni acción registrada, entonces no se podrá ejecutar pues no habrá ningún historial el cual mostrar al usuario.

5.8. Salir

Para la opción salir solamente le asignaremos un break para qué el programa finalice la ejecución del código entero finalizando así la compra del usuario.

Si el usuario digita alguna otra cosa que no sea un valor numérico o algún valor numérico fuera del rango de 1 a 8, entonces el menú saltará un error de interacción.

```
case 8:
    System.out.println("Saliste del menu principal");;
    break;
default :
    System.out.println("OPCION INVALIDA, DIGITE UN NUMERO DEL MENU");
```

Conclusión

A nivel de complejidad el desarrollo del programa es de nivel intermedio, pues se usan comandos y estructuras que a algunas personas (como es mi caso) tardan un poco de tiempo y bastante práctica para llegar al nivel de comprensión y lógica de programación, resulta ser bastante satisfactorio concluir con el desarrollo de este código pues se pone a prueba lo aprendido e incita a la investigación autodidacta para la resolución de problemas nuevos que llegan a presentarse con el paso del tiempo.