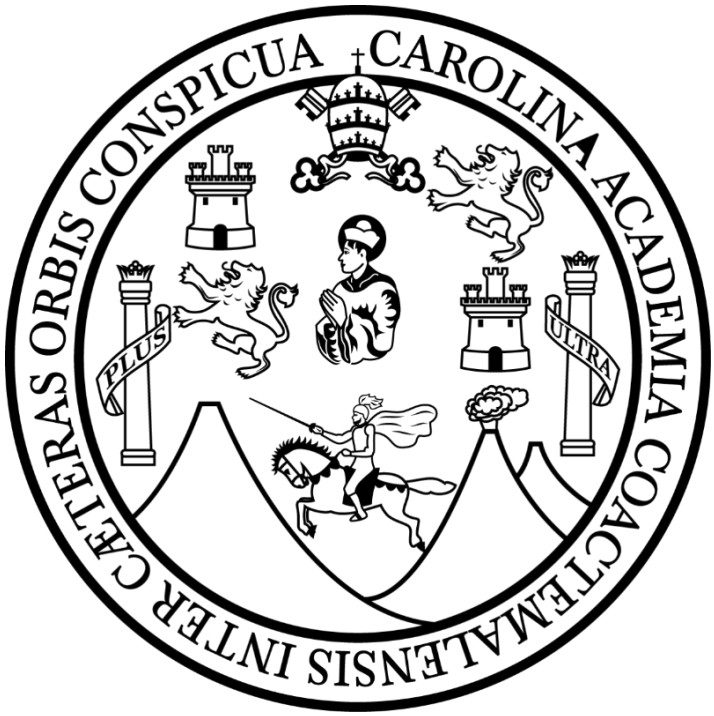
Universidad de San Carlos de Guatemala

Facultad de Ingeniería
Curso: Introducción a la Programación y Computación 1
Sección: F
Ing. William Estuardo Escobar Argueta

Tutor Académico: Zenaida Irazema Chacón García



Proyecto 1 - Manual de Desarrollo

Sistema de inventario tienda ropa

Nombre: Gerson Aaron Matta Aguilar

Carné: 202403711

Fecha: 13/09/2025

MANUAL DE DESARROLLO

19/08/2025

Problemas y soluciones encontradas:

- Validación de stock y precios negativos: no se validaba que el precio o el stock fueran números positivos al agregar un producto.
    - Se creó el método `validarPositivo()` que retorna -100 si el valor es negativo y muestra un mensaje de error.
    - Se agregó una validación en `agregarProducto()` para evitar generar un código único si los datos de stock y precios son inválidos.

Explicación:

- Se creó la estructura principal del proyecto, tales como las variables globales, la definición de los diferentes arreglos que serán de utilidad para el almacenamiento del programa.

```
public class Principal {

    //___ DECLARACION DE VARIABLES, ARRAYS, ETC. ___

    Scanner scanner = new Scanner(System.in);

    String nombre, categoria;
    double precio;
    int opcion, cantidadStock, codigoUnico;

    String[][] inventario = new String[100][5];
    String[][] ventas = new String[100][4];
    String[][] bitacora = new String[100][4];

    //_____________________________________________
```

- Se creó la función de `menu` que despliega las diferentes opciones del menú tanto visual como funcional:

```
public void menu(){
    do{
        System.out.println("\n+----------------------------------------+");
        System.out.println("|             MENÚ PRINCIPAL             |");
        System.out.println("+----------------------------------------+");
        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. Bitácora.                        |");
        System.out.println("|    8. Salir.                           |");
        System.out.println("+----------------------------------------+");
        opcion = ingresarEntero("*    Elige una opcion (1-8): ");
        switch(opcion){
            case 1:
                System.out.println("\n+----------------------------------------+");
                System.out.println("|            AGREGAR PRODUCTO            |");
                System.out.println("+----------------------------------------+");
                nombre = ingresarTexto("*    Nombre: ");
                // ... TODA LA DEMAS LOGICA DEL CASE
                break;
            case 2:
                do{
                    System.out.println("\n+----------------------------------------+");
                    System.out.println("|            BUSCAR  PRODUCTO            |");
                    System.out.println("+----------------------------------------+");
                    System.out.println("|    1. Buscar por código.               |");
                    System.out.println("|    2. Buscar por nombre.               |");
                    System.out.println("|    3. Buscar por categoría.            |");
                    System.out.println("|    4. Regresar         .               |");
                    System.out.println("+----------------------------------------+");
                    opcion = ingresarEntero("*    Elige una opcion (1-4): ");
                    switch(opcion){
                        // ... TODA LA DEMAS LOGICA DEL SWITCH Y LAS OPCIONES
                    }while(opcion!=4);

            case 3:
                break;
            case 4:
                break;
             // ... ETC
```

- Método para poder agregar un producto (faltando el generar el código único)

```
public void agregarProducto(String nombre, String categoria, double precio, int
cantidadStock){
    double confirmarPrecio = validarPositivo(1,precio);
    int confirmarStock = (int) validarPositivo(2,cantidadStock);
    codigoUnico = generarCodigoUnico();

    if( confirmarPrecio != -100 && confirmarStock != -100){
        for(int fila=0;fila<100;fila++){
            if(inventario[fila][0].equals("-100")){
                inventario[fila][0] = String.valueOf(codigoUnico);
                inventario[fila][1] = nombre;
                inventario[fila][2] = categoria;
                inventario[fila][3] = String.valueOf(precio);
                inventario[fila][4] = String.valueOf(cantidadStock);
                System.out.println("[+]    Producto agregado exitosamente.");
                break;
            }
        }
    }

}
```

- Método para buscar productos ya sea por código, nombre y categoría.

```
public void verProducto(int opcion, String textoIngresado){
    int contador = 0;
    switch(opcion){
        case 1:
            for (int fila=0;fila<100;fila++){
                if(inventario[fila][0].equals(String.valueOf(textoIngresado))){
                    System.out.println("-    Código: "+inventario[fila][0]);
                    System.out.println("-    Nombre: "+inventario[fila][1]);
                    System.out.println("-    Categoría: "+inventario[fila][2]);
                    System.out.println("-    Precio: "+inventario[fila][3]);
                    System.out.println("-    Stock: "+inventario[fila][4]);
                    contador++;
                    break;
                }
            }
            if(contador==0){
                System.out.println("[?]   NO SE HA ENCONTRADO NINGUN PRODUCTO.");
                break;
            }
        case 2:
            for (int fila=0;fila<100;fila++){
                if(inventario[fila][1].equals(String.valueOf(textoIngresado))){
                    System.out.println("-    Código: "+inventario[fila][0]);
                    System.out.println("-    Nombre: "+inventario[fila][1]);
                    System.out.println("-    Categoría: "+inventario[fila][2]);
                    System.out.println("-    Precio: "+inventario[fila][3]);
                    System.out.println("-    Stock: "+inventario[fila][4]);
                    contador++;
                }
            }
            if(contador==0){
                System.out.println("[?]   NO SE HA ENCONTRADO NINGUN PRODUCTO.");
                break;
            }
        case 3:
            for (int fila=0;fila<100;fila++){
                if(inventario[fila][2].equals(String.valueOf(textoIngresado))){
                    System.out.println("-    Código: "+inventario[fila][0]);
                    System.out.println("-    Nombre: "+inventario[fila][1]);
                    System.out.println("-    Categoría: "+inventario[fila][2]);
                    System.out.println("-    Precio: "+inventario[fila][3]);
                    System.out.println("-    Stock: "+inventario[fila][4]);
                    contador++;
                }
            }
            if(contador==0){
                System.out.println("[?]   NO SE HA ENCONTRADO NINGUN PRODUCTO.");
                break;
            }
    }
}
```

- Métodos auxiliares para la generación de códigos únicos y validar que los códigos sean únicos (aún no funcionales).

```
public int generarCodigoUnico(){
    return 1;
}

public int validarCodigoUnico(int codigoUnico){
    return 1;
}
```

- Métodos de validación tales como validar positivo, ingresar números enteros, decimales e ingresar una cadena de texto.

```
public double validarPositivo(int mensaje, double numeroIngresado){
    if(numeroIngresado>=0){
        return numeroIngresado;
    }else{
        if(mensaje==1){
            System.out.println("El precio no puede ser negativo.");
        }
        if(mensaje==2){
            System.out.println("La cantidad en Stock no puede ser negativo.");
        }
        if(mensaje==3){
            System.out.println("El número no puede ser negativo.");
        }
    }
    return -100;
}


public int ingresarEntero(String mensaje) {
    int num = 0;
    boolean valido = false;
    do {
        try {
            System.out.print(mensaje);
            String input = scanner.nextLine();
            num = Integer.parseInt(input);
            valido = true;
        } catch (NumberFormatException e) {
            System.out.println("[?]    Error: Debe ingresar un número entero válido. Porfavor
ingresar un número válido.");
        }
    } while (!valido);
    return num;
}
```

```java
public String ingresarTexto(String mensaje) {
    String texto = "";
    boolean valido = false;
    do {
        System.out.print(mensaje);
        texto = scanner.nextLine().trim();
        if (!texto.isEmpty() && texto.matches("[a-zA-ZáéíóúÁÉÍÓÚñÑ\\s]+")) {
            valido = true;
        } else {
            System.out.println("[?]    Error: Solo se permiten letras. Porfavor ingresar un texto
válido: ");
        }
    } while (!valido);
    return texto;
}


public double ingresarDecimal(String mensaje) {
    double decimal = 0;
    boolean valido = false;
    do {
        try {
            System.out.print(mensaje);
            String input = scanner.nextLine();
            decimal = Double.parseDouble(input);
            valido = true;
        } catch (NumberFormatException e) {
            System.out.println("[?]    Error: Debe ingresar un decimal válido. Porfavor ingresar un
número válido.");
        }
    } while (!valido);
    return decimal;
}
```

20/08/2025

Problemas y soluciones encontradas:

- Generación de códigos únicos no funcional: los métodos `generadorCodigoUnico()` y `validarCodigoUnico()` devolvían siempre 1, sin lógica real.
    - Se implementaron métodos autoincrementables:
        - `generarCodigoUnicoProducto()`
        - `generarCodigoUnicoVenta()`
        - `generarCodigoUnicoBitacora()`

  Se usaron variables globales (`ultimoCodigoProducto`, `ultimoCodigoVenta`, etc.) para llevar el conteo.

- Inicialización incorrecta del arreglo de `inventario`: los arreglos `inventario`, `ventas` y `bitácora` se inicializaban sin valores predeterminados, lo que podía causar errores al acceder a posiciones con valores `null`.
    - Se creó el método `generarEspaciosVacios()` (luego renombrado a `generarEspaciosVaciosInventario()`) para inicializar todos los espacios del arreglo con el valor "-100" o "VACIO".

  Esto evita errores de comparación y acceso a posiciones que aún no han sido inicializadas.

- Eliminación de productos sin reordenar el `inventario`: al eliminar un producto, quedaban espacios vacíos en medio del arreglo.
    - Se creó el método `ordenarProductos()` para compactar el arreglo después de una eliminación.

Explicación:

- Se implementó la funcionalidad de código único auto incrementable en `agregarProducto`, por lo que se creó un nuevo método llamado `generarCodigoUnico`, que es llamado y guardado dentro de una variable llamada `codigoUnico` para posteriormente poder guardarlo dentro de `inventario`.

```
public int generarCodigoUnicoProducto(){
    return ++ultimoCodigoProducto;
}
```

- Se agregó una parte de validación sobre si hay o no espacio disponible dentro de `inventario`, por lo tanto, si ya no se encuentran espacios disponibles, ya no habrá errores al intentar agregar un producto nuevo cuando no hay espacio suficiente.

```
boolean espacioDisponible = false;
for(int fila=0;fila<100;fila++){
    if(inventario[fila][0].equals("-100")){
        espacioDisponible=true;
        break;
    }
}

if(!espacioDisponible){
    System.out.println("[?]   INVENTARIO LLENO.");
    registrarBitacora(TIPOACCION_AGREGAR,ACCION_ERRONEA,usuario,"INGRESÓ A LA
OPCION 1, INVENTARIO LLENO");
    return;
}
        if(!espacioDisponible){
            System.out.println("+---------------------------------------+");
            System.out.println("[?]   INVENTARIO LLENO.");
            return;
        }
```

Se creó un método llamado `generarEspaciosVacios` que se ejecuta al iniciar el programa, con el objetivo de no tener valores `null` dentro del arreglo, esto con el objetivo de evitar diferentes tipos de errores.

```
public void generarEspaciosVacios(){
    for (int fila=0;fila<100;fila++){
        inventario[fila][0] = "-100";
        inventario[fila][1] = String.valueOf("VACIO");
        for (int col=2;col<5;col++){
            inventario[fila][col] = "VACIO";
        }
    }
}
```

- Se creó el método para eliminar un producto.

```
public void eliminarProducto(int codigo){
    boolean encontrado = false;
    for(int fila=0;fila<100;fila++){
        if(inventario[fila][0].equals(String.valueOf(codigo))){

            System.out.println("+---------------------------------------+");
            System.out.println("[✓]    ESTAS SEGURO DE ELIMINAR EL");
            System.out.println("       PRODUCTO "+inventario[fila][1]+", ");
            System.out.println("       CON CÓDIGO "+inventario[fila][0]+" ?");
            System.out.println("[X]    VUELVE A ESCRIBIR SU CODIGO");
            System.out.print("       PARA CONFIRMAR: ");
            codigoUnico = ingresarEntero("");

            if(inventario[fila][0].equals(String.valueOf(codigoUnico))){

                inventario[fila][0] = "-100";
                inventario[fila][1] = "VACIO";
                inventario[fila][2] = "VACIO";
                inventario[fila][3] = "VACIO";
                inventario[fila][4] = "VACIO";

                ordenarProductos();

                encontrado=true;
                System.out.println("+---------------------------------------
+");
                System.out.println("[-]    PRODUCTO ELIMINADO EXITOSAMENTE.");
                break;
            }

            System.out.println("+---------------------------------------+");
            System.out.println("[X]    ACCION CANCELADA.");
            return;

        }
    }
    if(!encontrado){
        System.out.println("+---------------------------------------+");
        System.out.println("[?]   NO SE HA ENCONTRADO NINGUN PRODUCTO");
        System.out.println("      CON ESE CÓDIGO.");
        return;
    }
}
```

- Se creó el método `ordenarProductos` que es llamado dentro del método de `eliminarProducto` con el objetivo de que, al eliminar un producto, la posición de este no quede vacía, por lo tanto, gracias a este método se borra este espacio vacío y se corren todas las demás posiciones una posición adelante, haciendo que todos los espacios ocupados siempre se encuentren arriba y las posiciones vacías se encuentren de último en el arreglo.

```
public void ordenarProductos(){
    String[][] nuevoInventario = new String[100][5];
    int contador = 0;
    for(int fila=0;fila<100;fila++){
        if(!(inventario[fila][0].equals("-100"))){
            nuevoInventario[contador][0] = inventario[fila][0];
            nuevoInventario[contador][1] = inventario[fila][1];
            nuevoInventario[contador][2] = inventario[fila][2];
            nuevoInventario[contador][3] = inventario[fila][3];
            nuevoInventario[contador][4] = inventario[fila][4];
            contador++;
        }
    }
    for(int fila=contador;fila<100;fila++){
        nuevoInventario[fila][0] = "-100";
        nuevoInventario[fila][1] = "VACIO";
        nuevoInventario[fila][2] = "VACIO";
        nuevoInventario[fila][3] = "VACIO";
        nuevoInventario[fila][4] = "VACIO";
    }
    for(int fila=0;fila<100;fila++){
        inventario[fila][0]=nuevoInventario[fila][0];
        inventario[fila][1]=nuevoInventario[fila][1];
        inventario[fila][2]=nuevoInventario[fila][2];
        inventario[fila][3]=nuevoInventario[fila][3];
        inventario[fila][4]=nuevoInventario[fila][4];
    }

}
```

- Se creó el método para mostrar los datos del estudiante.

```
public void mostrarDatosEstudiante(){
    System.out.println("\n+----------------------------------------+");
    System.out.println("|          DATOS DEL ESTUDIANTE          |");
    System.out.println("+----------------------------------------+");
    System.out.println("|    Nombre: Gerson Aarón Matta Aguilar  |");
    System.out.println("|    Carnet: 202403711                   |");
    System.out.println("|    Curso: Introducción a la            |");
    System.out.println("|           Programacion y Computacion 1 |");
    System.out.println("|    Sección: F                          |");
    System.out.println("+----------------------------------------+");
}
```

21/08/2025

Problemas y soluciones encontradas:

- Manejo de productos repetidos en `ventas`: al registrar una venta, no se validaba si un producto ya estaba en el `carrito`, lo que causaba duplicados.
    - Se agregó lógica para buscar productos repetidos en el `carrito` y actualizar la cantidad y el subtotal en lugar de duplicar la entrada.

Explicación:

- Se agregó el método de `registrarVenta`, con todas sus validaciones de existencia, stock y restar las unidades vendidas al `inventario`, además también valida si el producto ya está agregado simplemente modifica la cantidad y el subtotal mediante:

```
for(int fila=0;fila<100;fila++){

if(Integer.parseInt(inventario[existeId][0])==Integer.parseInt(carrito[fila][0])
){
        int tempCantidad = cantidad+Integer.parseInt(carrito[fila][2]);
        if(validarStock(existeId, tempCantidad)){

            monto = Integer.parseInt(inventario[existeId][3])*cantidad;
            total = total + monto;
            cantidad = cantidad+Integer.parseInt(carrito[fila][2]);
            monto = Integer.parseInt(inventario[existeId][3])*cantidad;

            carrito[fila][2] = String.valueOf(cantidad);
            carrito[fila][3] = String.valueOf(monto);
            repetido = true;

            System.out.println("[+]    PRODUCTO AGREGADO AL CARRITO");
            System.out.println("       EXITOSAMENTE");
        }
        break;
    }
}
```

De lo contrario simplemente agregará el nuevo producto al arreglo de `carrito` ejecutando lo siguiente:

```
if(repetido==false && validarStock(existeId, cantidad)){ //Agrega una nueva
columna al producto nuevo.

    monto = Integer.parseInt(inventario[existeId][3])*cantidad;
```

```
    total = total + monto;

    carrito[contador][0] = inventario[existeId][0];
    carrito[contador][1] = inventario[existeId][1];
    carrito[contador][2] = String.valueOf(cantidad);
    carrito[contador][3] = String.valueOf(monto);

    System.out.println("[+]    PRODUCTO AGREGADO AL CARRITO");
    System.out.println("       EXITOSAMENTE");

    contador++;
}
```

- Cuando el usuario haya agregado todos los productos que desea, al seleccionar la opción 2, ejecutará el caso 2 del switch haciendo que se reste el stock en `inventario` y que se registre la venta en el arreglo de `ventas` con los distintos detalles mediante:

```
LocalDateTime fecha = LocalDateTime.now();
DateTimeFormatter fecha2 = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
String fechaFormateada = fecha.format(fecha2);
int posicion = generarCodigoUnicoVenta();
String cantProd="";

for(int fila=0;fila<100;fila++){
    if(!carrito[fila][0].equals("-100")){
        int posicionProducto =
buscarProducto(Integer.parseInt(carrito[fila][0]));
        cantProd=cantProd+carrito[fila][0]+","+carrito[fila][2]+"|";
        inventario[posicionProducto][4] =
String.valueOf(Integer.parseInt(inventario[posicionProducto][4])-
Integer.parseInt(carrito[fila][2]));
    }
}
ventas[posicion][0] = cantProd;
ventas[posicion][1] = fechaFormateada;
ventas[posicion][2] = String.valueOf(total);

System.out.println("\n+----------------------------------------+");
System.out.println("|       LA VENTA HA SIDO REGISTRADA      |");
System.out.println("+----------------------------------------+");
opcion=3;
```

- Para el funcionamiento del método de `registrarVenta` se implementaron otros métodos auxiliares tales como: `validarStock` y `buscarProducto` (retorna la posición del array en donde se encuentra ese producto)

```
public boolean validarStock(int codProdInventario, int cantidadVenta){
    if(Integer.parseInt(inventario[codProdInventario][4])<cantidadVenta){
        System.out.println("[X]    STOCK INSUFICIENTE PARA: "+cantidadVenta+"
UNIDADES,");
        System.out.println("       CONTAMOS CON:
"+inventario[codProdInventario][4]+" UNIDADES.");
        return false;
    }
    return true;
}

public int buscarProducto(int codigo){
    for (int fila=0;fila<100;fila++){
        if(inventario[fila][0].equals(String.valueOf(codigo))){
            return fila;
        }
    }
    return -100;
}
```

- También se realizó un cambio en el nombre del método `generarCodigoUnico` a `generarCodigoUnicoProducto`, debido a que ahora hay otro nuevo método que genera el código único de ventas (`generarCodigoUnicoVenta`), por lo tanto se hizo este cambio para evitar confusiones.

```
public int generarCodigoUnico (){
public int generarCodigoUnicoProducto(){
    return ++ultimoCodigoProducto;
}

public int generarCodigoUnicoVenta(){
    return ++ultimoCodigoVenta;
}
```

- Lo mismo para `generarEspaciosVacios`, ahora es `generarEspaciosVaciosInventario`, ya que ahora existe también `generarEspaciosVaciosCarrito`.

23/08/2025

Problemas y soluciones encontradas:

- Falta de registros en `bitácora` para acciones críticas: se necesitaba que las acciones registraban en el arreglo de la `bitácora`.
    - Se agregaron llamadas a `registrarBitacora()` en todos los métodos principales (agregarProducto, buscar, eliminar, ventas, etc.), tanto para acciones exitosas como fallidas.

- Formato de fechas inconsistente: la fecha se generaba de manera redundante en múltiples métodos.
    - Se centralizó la generación de fechas en el método `generarFechaHoraActual()`.

- Errores de parseo en cálculos de ventas: se usaba `Integer.parseInt()` para precios (que son decimales), lo que causaba errores de conversión.
    - Se reemplazó por `Double.parseDouble()` en los cálculos de monto y total.

- Búsqueda ineficiente de productos: el método `verProducto()` no terminaba correctamente después de encontrar un producto (faltaban break en los case).
    - Se agregaron break en cada caso del switch.

Explicación:

- Se agregó el método de `registrarBitacora`, la cual su objetivo es el registro de cada acción, estas se almacenan en el vector de `bitácora`, guardando la fecha, detalles de las acciones, que usuario y el detalle de lo que se realizó (mensaje).

```
public void registrarBitacora(String tipoAccion, String accion, String usuario,
String mensaje){
    String fecha = generarFechaHoraActual();
    int posicion = generarCodigoUnicoBitacora();
    bitacora[posicion][0]= fecha;
    bitacora[posicion][1]= tipoAccion;
    bitacora[posicion][2]= accion;
    bitacora[posicion][3]= usuario;
    bitacora[posicion][4]= mensaje;
}
```

- Se creo el método `generarCodigoUnicoBitacora`, el cual es llamado en el método de `registrarBitacora`.

```
public int generarCodigoUnicoBitacora(){
    return ++ultimoCodigoBitacora;
}
```

- Se creo el método `generarFechaHoraActual`, el cual es llamado en el método de `registrarBitacora` y que servirá para funciones posteriores.

```
public String generarFechaHoraActual(){
    LocalDateTime fecha = LocalDateTime.now();
    DateTimeFormatter fecha2 = DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm:ss");
    String fechaFormateada = fecha.format(fecha2);
    return fechaFormateada;
}
```

- Se crearon nuevas variables globales para guardar datos del usuario (se habla en el inciso de abajo), y para guardar e indicar el tipo de acciones de la bitácora.

```
public class Principal {

    //___ DECLARACION DE VARIABLES, ARRAYS, ETC. ___

    Scanner scanner = new Scanner(System.in);

    String nombreUsuario;
    String carnetUsuario;
    String usuario;

    String nombre, categoria;
    double precio;
    int opcion, cantidadStock, codigoUnico, cantidad;

    int ultimoCodigoProducto=-1,ultimoCodigoVenta = -1,ultimoCodigoBitacora=-1;

    String[][] inventario = new String[100][5];
    String[][] ventas = new String[100][3];
    String[][] bitacora = new String[200][5];
    String nombreUsuario;
    String carnetUsuario;
    String usuario;

    //ACCIONES DE LA BITACORA
    String TIPOACCION_AGREGAR = "AGREGAR";
    String TIPOACCION_BUSCAR = "BUSCAR";
    String TIPOACCION_ELIMINAR = "ELIMINAR";
    String TIPOACCION_REGISTRAR = "REGISTRAR";
    String TIPOACCION_GENERAR = "GENERAR";
    String TIPOACCION_VER = "VER";

    String ACCION_CORRECTA = "CORRECTA";
    String ACCION_ERRONEA = "ERRONEA";
```

- Para la `bitácora` se requiere un usuario para saber que persona es la que está realizando cierta acción. Por lo que ahora cada vez que se ejecuta el programa este solicitará un nombre y carnet, estos datos son los que se guardarán en la parte de usuario de la `bitácora` (`bitacora[posicion][3]= usuario`).

```
public void menu(){
    System.out.println("\n+----------------------------------------+");
    System.out.println("|           INGRESA TU NOMBRE            |");
    System.out.println("+----------------------------------------+");
    nombreUsuario = ingresarTexto("*    Nombre: ");
    System.out.println("\n+----------------------------------------+");
    System.out.println("|           INGRESA TU CARNET            |");
    System.out.println("+----------------------------------------+");
    carnetUsuario = String.valueOf(ingresarEntero("*    Carnet: "));
    System.out.println("+----------------------------------------+");
    System.out.println("[✓]    DATOS DEL USUARIO GUARDADOS.");
    usuario = nombreUsuario+","+carnetUsuario;
    System.out.println("+----------------------------------------+");
    do{
        System.out.println("\n+----------------------------------------+");
        System.out.println("|             MENÚ PRINCIPAL             |");
        System.out.println("+----------------------------------------+");
        // ... continuación del código de menú
```

- Ahora que el método de `registrarBitacora` está creado, llamo el método a cada uno de los otros diferentes métodos que realizan una acción dentro del programa (agregar, buscar y eliminar producto, registrar ventas, ver datos del estudiante y posteriormente para generar reportes y ver la bitácora) para registrar en la `bitácora` cada de una de las acciones.

```
public void registrarVenta() {

    // ... TODA LA DEMAS LOGICA DEL METODO

    System.out.println("\n+----------------------------------------+");
    System.out.println("|       LA VENTA HA SIDO REGISTRADA      |");
    System.out.println("+----------------------------------------+");
    registrarBitacora(TIPOACCION_REGISTRAR,ACCION_CORRECTA,usuario,"PRODUCTOS:
"+cantProd+"    "+"TOTAL: "+total );
    opcion=3;

        }else{
            System.out.println("+----------------------------------------+");
            System.out.println("[?]   NO SE HA AGREGADO NINGUN PRODUCTO.");
            System.out.println("+----------------------------------------+");
            registrarBitacora(TIPOACCION_REGISTRAR,ACCION_ERRONEA,usuario,"SE
INTENTO REGISTRAR VENTA SIN PRODUCTOS AGREGADOS");
        }
```

```
public void eliminarProducto() {

    // ... TODA LA DEMAS LOGICA DEL METODO

    System.out.println("[-]    PRODUCTO ELIMINADO EXITOSAMENTE.");

    registrarBitacora(TIPOACCION_ELIMINAR,ACCION_CORRECTA,usuario,"CODIGO:
"+codigo);
    break;
    }

    System.out.println("+--------------------------------------+");
    System.out.println("[X]    ACCION CANCELADA.");
    registrarBitacora(TIPOACCION_ELIMINAR,ACCION_ERRONEA,usuario,"EL USUARIO
CANCELO LA ELIMINACION DEL PRODUCTO CON CODIGO: "+codigo);
    return;
```

Y así vamos realizando lo mismo con cada uno de los métodos ya creados.

- `public void verProducto()`
- `public void agregarProducto()`
- `public void mostrarDatosEstudiante()`

- Corrección de un error que al ingresar un stock o monto negativo se generaba el código único igualmente, haciendo que se generara ese código sin razón provocando que ya no se utilizara para posteriores nuevos productos.

```
public void agregarProducto(String nombre, String categoria, double precio, int
cantidadStock){

    // ... TODA LA DEMAS LOGICA DEL METODO

    double confirmarPrecio = validarPositivo(1,precio);
    int confirmarStock = (int) validarPositivo(2,cantidadStock);
    codigoUnico = generarCodigoUnicoProducto();

    if( confirmarPrecio != -100 && confirmarStock != -100){

        codigoUnico = generarCodigoUnicoProducto();

        for(int fila=0;fila<100;fila++){
            if(inventario[fila][0].equals("-100")){
                inventario[fila][0] = String.valueOf(codigoUnico);
                inventario[fila][1] = nombre;
                inventario[fila][2] = categoria;
                inventario[fila][3] = String.valueOf(precio);
                inventario[fila][4] = String.valueOf(cantidadStock);
                System.out.println("+----------------------------------------+");
                System.out.println("[+]    PRODUCTO AGREGADO EXITOSAMENTE.");
                System.out.println("       CON CÓDIGO: "+codigoUnico);
                break;
                registrarBitacora(TIPOACCION_AGREGAR,ACCION_CORRECTA,usuario,"CÓDIGO:
"+codigoUnico);
                return;
            }
        }
    }
    registrarBitacora(TIPOACCION_AGREGAR,ACCION_ERRONEA,usuario,"NÚMERO INGRESADO DE
PRECIO/STOCK INVÁLIDO");

}
```

- Corrección de un error a la hora de buscar productos, falta colocar break al final de cada caso para que no se registraran las tres acciones a la vez en la bitácora, además se optimizó el obtener la fecha actual llamando simplemente al método `generarFechaHoraActual`.

```
public void verProducto(int opcion, String textoIngresado){
    int contador = 0;
    switch(opcion){
        case 1:

          // ... TODA LA DEMAS LOGICA DEL METODO

            if(contador==0){
                System.out.println("[?]   NO SE HA ENCONTRADO NINGUN PRODUCTO.");
                registrarBitacora(TIPOACCION_BUSCAR,ACCION_ERRONEA,usuario,"NINGUN
PRODUCTO CON CODIGO: "+textoIngresado);
                break;
            }
            break;
        case 2:

          // ... TODA LA DEMAS LOGICA DEL METODO

            if(contador==0){
                System.out.println("[?]   NO SE HA ENCONTRADO NINGUN PRODUCTO.");
                registrarBitacora(TIPOACCION_BUSCAR,ACCION_ERRONEA,usuario,"NINGUN
PRODUCTO CON NOMBRE: "+textoIngresado);
                break;
            }
            registrarBitacora(TIPOACCION_BUSCAR,ACCION_CORRECTA,usuario,"NOMBRE:
"+textoIngresado);
            break;
        case 3:

          // ... TODA LA DEMAS LOGICA DEL METODO

            if(contador==0){
                System.out.println("[?]   NO SE HA ENCONTRADO NINGUN PRODUCTO.");
                registrarBitacora(TIPOACCION_BUSCAR,ACCION_ERRONEA,usuario,"NINGUN
PRODUCTO CON CATEGORIA: "+textoIngresado);
                break;
            }
            registrarBitacora(TIPOACCION_BUSCAR,ACCION_CORRECTA,usuario,"CATEGORIA:
"+textoIngresado);
            break;
    }
}
```

- Corrección de un error a la hora de validar el límite de registrar venta ya que simplemente no se estaba validando tal opción y también se corrigió el error al agregar ya un producto repetido, ya que se estaban parseando a Enteros en vez de a Double.

```
public void registrarVenta() {

        // ... TODA LA DEMAS LOGICA DEL METODO

        int tempCantidad = cantidad + Integer.parseInt(carrito[fila][2]);

        if(validarStock(existeId, tempCantidad)){

            monto = Integer.parseInt(inventario[existeId][3])*cantidad;
            monto = Double.parseDouble(inventario[existeId][3])*cantidad;
            total = total + monto;
            cantidad = cantidad+Integer.parseInt(carrito[fila][2]);
            monto = Integer.parseInt(inventario[existeId][3])*cantidad;
            monto = Double.parseDouble(inventario[existeId][3])*cantidad;

            carrito[fila][2] = String.valueOf(cantidad);
            carrito[fila][3] = String.valueOf(monto);
            repetido = true;

            System.out.println("[+]    PRODUCTO AGREGADO AL CARRITO");
            System.out.println("       EXITOSAMENTE");
        }
        break;
    }
}
            if(repetido==false && validarStock(existeId, cantidad)){

                monto = Integer.parseInt(inventario[existeId][3])*cantidad;
                monto = Double.parseDouble(inventario[existeId][3])*cantidad;
                total = total + monto;

                carrito[contador][0] = inventario[existeId][0];
                carrito[contador][1] = inventario[existeId][1];
                carrito[contador][2] = String.valueOf(cantidad);
                carrito[contador][3] = String.valueOf(monto);

                System.out.println("[+]    PRODUCTO AGREGADO AL CARRITO");
                System.out.println("       EXITOSAMENTE");
                contador++;
            }
            break;
        }else{
            System.out.println("[?]   NO SE HA ENCONTRADO NINGUN PRODUCTO");
            System.out.println("      CON ESE CÓDIGO.");
        }
        break;
```

```
    case 2:
        if(total!=0){
            LocalDateTime fecha = LocalDateTime.now();
            DateTimeFormatter fecha2 = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
            String fechaFormateada = fecha.format(fecha2);
            String fecha = generarFechaHoraActual();
            int posicion = generarCodigoUnicoVenta();

            if(posicion>200){
                System.out.println("+----------------------------------------+");
                System.out.println("[?]   SE HA ALCANZADO EL CUPO MAXIMO DE ");
                System.out.println("      VENTAS PARA ALMACENAR.             ");
                System.out.println("+----------------------------------------+");
                registrarBitacora(TIPOACCION_REGISTRAR,ACCION_ERRONEA,usuario,"ESPACIO PARA
ALMACENAR VENTAS ESTA LLENO");
                break;
            }

            String cantProd="";

            for(int fila=0;fila<100;fila++){
                if(!carrito[fila][0].equals("-100")){
                    int posicionProducto = buscarProducto(Integer.parseInt(carrito[fila][0]));
                    cantProd=cantProd+carrito[fila][0]+","+carrito[fila][2]+"|";
                    inventario[posicionProducto][4] =
String.valueOf(Integer.parseInt(inventario[posicionProducto][4])-
Integer.parseInt(carrito[fila][2])); //Restar del inventario
                }
            }
            ventas[posicion][0] = cantProd;
            ventas[posicion][1] = fechaFormateada;
            ventas[posicion][1] = fecha;
            ventas[posicion][2] = String.valueOf(total);

            System.out.println("\n+----------------------------------------+");
            System.out.println("|       LA VENTA HA SIDO REGISTRADA      |");
            System.out.println("+----------------------------------------+");
            registrarBitacora(TIPOACCION_REGISTRAR,ACCION_CORRECTA,usuario,"PRODUCTOS: "+cantProd+"
"+"TOTAL: "+total );
            opcion=3;

        }else{
            System.out.println("+----------------------------------------+");
            System.out.println("[?]   NO SE HA AGREGADO NINGUN PRODUCTO.");
            System.out.println("+----------------------------------------+");
            registrarBitacora(TIPOACCION_REGISTRAR,ACCION_ERRONEA,usuario,"SE INTENTO REGISTRAR
VENTA SIN PRODUCTOS AGREGADOS");
        }

        break;
    case 3:
        break;
```

30/08/2025

Problemas y soluciones encontradas:

- `Bitácora` sin validación de límites: no se validaba si la `bitácora` tenía espacio disponible antes de registrar una acción.
    - Se agregó una validación en `registrarBitacora()` para evitar escribir si el arreglo está lleno.

Explicación:

- Se creó el método de para la generación de reportes del `inventario` (Stock) mediante un PDF, para el funcionamiento de este se agregó la librería de `itextpdf` con sus debidas importaciones.

```
public void generarPdfStock(){

    LocalDateTime fechaSinFormato = LocalDateTime.now();
    String fechaArchivo =
fechaSinFormato.format(DateTimeFormatter.ofPattern("dd_MM_YYYY_HH_mm_ss"));
    String fechaTexto =fechaSinFormato.format(DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss"));

    String rutaArchivo = "C:\\Users\\TheSn\\Downloads\\"+fechaArchivo+"_Stock.pdf";
    Document documento = new Document();

    try {

        PdfWriter.getInstance(documento, new FileOutputStream(rutaArchivo));

        documento.open();

        // Estilos
        Font tituloFont = FontFactory.getFont(FontFactory.HELVETICA_BOLD, 18, BaseColor.DARK_GRAY);
        Font textoFont1 = FontFactory.getFont(FontFactory.HELVETICA, 9, BaseColor.GRAY);
        Font textoFont2 = FontFactory.getFont(FontFactory.HELVETICA_BOLD, 14, BaseColor.RED);
        Font encabezadoFont = FontFactory.getFont(FontFactory.HELVETICA_BOLD, 12, BaseColor.WHITE);
        Font celdaFont = FontFactory.getFont(FontFactory.HELVETICA, 10);

        // Titulo/Informacion
        Paragraph titulo = new Paragraph("REPORTE DE INVENTARIO - STOCK", tituloFont);
        titulo.setAlignment(Element.ALIGN_CENTER);
        documento.add(titulo);

        Paragraph informacion = new Paragraph(
```

```
            "Generado el: "+fechaTexto+"    |    Usuario: "+nombreUsuario+"    |    Carnet:
"+carnetUsuario, textoFont1
        );
        informacion.setAlignment(Element.ALIGN_CENTER);
        informacion.setSpacingAfter(15);
        documento.add(informacion);

        // Crear la tabla
        PdfPTable tabla = new PdfPTable(5);
        float[] anchosColumnas = {1f, 3f, 2f, 1.5f, 1.5f};
        tabla.setWidthPercentage(100);
        tabla.setWidths(anchosColumnas);

        // Encabezados
        PdfPCell h1 = new PdfPCell(new Paragraph("ID", encabezadoFont));
        h1.setBackgroundColor(new BaseColor(36, 48, 166));
        h1.setHorizontalAlignment(Element.ALIGN_CENTER);
        h1.setBorderWidth(1.2f);
        h1.setPadding(8);
        tabla.addCell(h1);

        // ... TODOS LOS DEMAS ENCABEZADOS

        int contarFilas = 0;

        // Agregar cada fila y columna
        for(int fila=0;fila<100;fila++){
            if(!inventario[fila][0].equals("-100")){
                contarFilas++;

                PdfPCell celdaCodigo = new PdfPCell(new Phrase(inventario[fila][0], celdaFont));
                celdaCodigo.setHorizontalAlignment(Element.ALIGN_LEFT);
                celdaCodigo.setPadding(5);
                tabla.addCell(celdaCodigo);

                // ... TODAS LAS DEMAS CELDAS (NOMBRE, CATEGORIA, PRECIO, STOCK)
            }
        }

        // Añadir la tabla al documento
        documento.add(tabla);

        if(contarFilas==0){
            Paragraph sinProductos = new Paragraph("NO HAY PRODUCTOS AGREGADOS EN EL INVENTARIO.",
textoFont2);
            sinProductos.setSpacingAfter(15);
            sinProductos.setAlignment(Element.ALIGN_CENTER);
            documento.add(sinProductos);
        }

        System.out.println("Tabla generada con éxito.");
        System.out.println("PDF creado en la siguiente ruta: " + rutaArchivo);
        registrarBitacora(TIPOACCION_GENERAR,ACCION_CORRECTA,usuario,"PDF STOCK GENERADO");
```

```
    } catch (Exception e) {
        e.printStackTrace();
        registrarBitacora(TIPOACCION_GENERAR,ACCION_ERRONEA,usuario,"ERROR AL GENERAR PDF STOCK");
    } finally {
        // Cerrar documento
        if (documento.isOpen()) {
            documento.close();
        }
    }
}
```

- Corrección de un error/validación a la hora de registrar algo en la `bitacora`, en donde si es mayor del espacio del array simplemente no se agrega.

```
public void registrarBitacora(String tipoAccion, String accion, String usuario,
String mensaje){
    String fecha = generarFechaHoraActual();
    int posicion = generarCodigoUnicoBitacora();
    if(!(posicion>200)){
        bitacora[posicion][0]= fecha;
        bitacora[posicion][1]= tipoAccion;
        bitacora[posicion][2]= accion;
        bitacora[posicion][3]= usuario;
        bitacora[posicion][4]= mensaje;
    }
}
```

- Se agregó el método para listar/ver todas las bitácoras registradas, este método se llama en el caso 7 del switch del método menú.

```
public void verListadoBitacora(){
    int contarBitacoras = 0;
    System.out.println("\n+----------------------+----------------+-----------
+----------------------+-------------------------------------------------------+");
    System.out.println("| FECHA Y HORA         | TIPO DE ACCION  | ACCION     |
USUARIO              | MENSAJE                                                |");
    System.out.println("+----------------------+----------------+-----------+-
--------------------+-------------------------------------------------------
-----------------------------+");
    for (int fila=0;fila<200;fila++){
        if(bitacora[fila][0]!=null){
            System.out.printf("| %-20.20s | %-15.15s | %-10.10s | %-20.20S | %-
85.85S | %n",
                bitacora[fila][0],
                bitacora[fila][1],
                bitacora[fila][2],
                bitacora[fila][3],
                bitacora[fila][4]
            );
            contarBitacoras++;
        }
    }
    if(contarBitacoras == 0){
        System.out.println("+----------------------+----------------+----------
--+----------------------+-------------------------------------------------------+");
        System.out.println("NO HAY REGISTROS.");
    }else{
        System.out.println("+----------------------+----------------+----------
--+----------------------+-------------------------------------------------------+");
        System.out.println("Bitacoras agregadas: ["+contarBitacoras+"/200]");
    }
}
```

31/08/2025

Problemas y soluciones encontradas:

- Decimales no redondeados en precios: el método `ingresarDecimal()` permitía cualquier cantidad de decimales, afectando la legibilidad de reportes.
    - Se agregó Math.round(decimal * 100.0) / 100.0 para redondear a 2 decimales.

Explicación:

- Se creó el método de para la generación de reportes de ventas mediante un PDF, siguiendo la misma lógica que el reporte de `inventario`, solo que ahora recorre el arreglo de `ventas`.

  Cambios importantes en este nuevo método (en lo demás se mantiene exactamente igual o simplemente cambian los nombres de las columnas o tamaños etc…):

```
// Agregar cada fila y columna
for(int fila=0;fila<100;fila++){
    if(ventas[fila][0]!=null){

        String textoC1 = "";
        String[] codigoCantidadVendida = ventas[fila][0].split("\\|");

        int tamañoCantVend = codigoCantidadVendida.length;

        for(int fila2=0;fila2<tamañoCantVend;fila2++){
            String[] detalleTamañoCantVend =
codigoCantidadVendida[fila2].split(",");

            textoC1=textoC1+
                    "Código: "+detalleTamañoCantVend[0]+",    "+
                    "Nombre: "+detalleTamañoCantVend[1]+",    "+
                    "Cantidad: "+detalleTamañoCantVend[2]+"\n";
        }

        PdfPCell celdaCodigoNombreCantidad = new PdfPCell(new Phrase(textoC1,
celdaFont));
        celdaCodigoNombreCantidad.setHorizontalAlignment(Element.ALIGN_LEFT);
        celdaCodigoNombreCantidad.setPadding(5);
        tabla.addCell(celdaCodigoNombreCantidad);

        PdfPCell celdaFechaHora = new PdfPCell(new Phrase(ventas[fila][1],
celdaFont));
        celdaFechaHora.setHorizontalAlignment(Element.ALIGN_LEFT);
        celdaFechaHora.setPadding(5);
        tabla.addCell(celdaFechaHora);
```

```
        PdfPCell celdaTotal = new PdfPCell(new Phrase("Q "+ventas[fila][2],
celdaFont));
        celdaTotal.setHorizontalAlignment(Element.ALIGN_CENTER);
        celdaTotal.setPadding(5);
        tabla.addCell(celdaTotal);

        contarFilas++;

    }
}
```

- Hubo una pequeña modificación en una línea del método ingresarDecimal, ahora este siempre aproxima a 2 decimales sin importar que tantos decimales ingrese el usuario, esto con el objetivo de evitar fallos visuales/impresión o simplemente evitar información irrelevante con tantos decimales que no llegan a importar a la hora de ver los precios o el total en las ventas.

```
public double ingresarDecimal(String mensaje) {
    double decimal = 0;
    boolean valido = false;

    do {
        try {
            System.out.print(mensaje);
            String input = scanner.nextLine();
            decimal = Double.parseDouble(input);
            decimal = Math.round(decimal * 100.0) / 100.0; //Redondear
            valido = true;
        } catch (NumberFormatException e) {
            System.out.println("[?]    ERROR: Debe ingresar un decimal válido.
Porfavor ingresar un número válido.");
        }
    } while (!valido);

    return decimal;
}
```

- Pequeñas modificaciones en los detalles/mensajes en la parte de la bitácora. Ahora se guardan de la siguiente forma:

```
registrarBitacora(TIPOACCION_AGREGAR,ACCION_ERRONEA,usuario,"INVENTARIO LLENO");
registrarBitacora(TIPOACCION_AGREGAR,ACCION_ERRONEA,usuario,"INGRESÓ A LA OPCION 1, INVENTARIO
LLENO");
```

```
registrarBitacora(TIPOACCION_BUSCAR,ACCION_CORRECTA,usuario,"CATEGORIA: "+textoIngresado);
registrarBitacora(TIPOACCION_BUSCAR,ACCION_CORRECTA,usuario,"INGRESÓ A LA OPCION 2, CATEGORIA:
"+textoIngresado);

// ... ETC ...
```

- Se modificó y se agregó nombre del producto en el arreglo de ventas, por lo que ahora se guarda el código, nombre y cantidad vendida.

```
public void registrarVenta() {
     // ... TODA LA DEMAS LOGICA DEL METODO
    case 2:
        if(total!=0){
            for(int fila=0;fila<100;fila++){
                if(!carrito[fila][0].equals("-100")){
                    int posicionProducto = buscarProducto(Integer.parseInt(carrito[fila][0]));
                    cantProd=cantProd+carrito[fila][0]+","+carrito[fila][2]+"|";
    cantProd=cantProd+carrito[fila][0]+","+inventario[posicionProducto][1]+","+carrito[fila][2]+"|";
                    inventario[posicionProducto][4] =
String.valueOf(Integer.parseInt(inventario[posicionProducto][4])-
Integer.parseInt(carrito[fila][2])); //Restar del inventario
                }
            }
     // ... TODA LA DEMAS LOGICA DEL METODO
}
```

01/09/2025

Problemas y soluciones encontradas:

- Categorías libres con problemas de duplicados: las `categorías` se ingresaban como texto libre, lo que podía generar inconsistencias como: Categorías: "Camisetas" y "Camisetas" que son dos totalmente distintas.
    - Se definió un arreglo preestablecido de `categorías`.
    - Se creó el método `ingresarValidarCategorias()` para que el usuario elija una opción numérica.

Explicación:

- Se modificó el funcionamiento de categorías, ahora son categorías ya preestablecidas, en donde el usuario debe de elegir alguna de esas opciones.

  Por lo tanto, se creó una nueva variable global de tipo arreglo que va almacenar las categorías.

```
String[] categorias = {
    "Camisetas y blusas", "Pantalones y jeans", "Faldas y vestidos", "Chamarras y abrigos", "Suéteres y sudaderas",
    "Ropa interior", "Calzado", "Accesorios", "Ropa deportiva", "Ropa de dormir"
};
```

  Se creó un nuevo método que funciona tanto para agregar producto como para buscar un producto por categoría llamado `ingresarValidarCategorias`.

```
public int ingresarValidarCategorias(){
    int categoriaIngresada;
    do{
        for(int i=0;i<categorias.length;i++){
            System.out.println("      "+(i+1)+". "+categorias[i]);
        }
        categoriaIngresada = ingresarEntero("*    Categoría: ");
        if(!(categoriaIngresada<=(categorias.length) && categoriaIngresada>0)){
            System.out.println("\n[?]    ERROR: Por favor ingrese un número entre 1 y " + categorias.length);
        }
    }while(!(categoriaIngresada<=(categorias.length) && categoriaIngresada>0));
    return categoriaIngresada;
}
```

  Se cambio la forma en la que se pide la categoría.

```
categoria = ingresarTexto("*    Categoría: ");
categoria = categorias[ingresarValidarCategorias()];
```