Manual de Sistema: Software de Control de Ventas y Compras

para Almacenes La Prosperidad

Índice

1. Introducción .
2. Manejo de bodega.
3. Manejo de ventas.
4. Devoluciones.
5. Cliente frecuentes.
6. Reportes diarios.
7. Contacto

INTRODUCCIÓN

1

El software desarrollado para Almacenes **La Prosperidad** tiene como objetivo principal modernizar el control de inventarios, ventas y devoluciones, conservando las funcionalidades esenciales que el almacén ha venido utilizando, mientras se implementan mejoras en eficiencia, manejo de información y procesos de venta.

El sistema ha sido desarrollado en **Java**, utilizando el entorno de desarrollo **NetBeans**. Esto permite que el sistema sea compatible con cualquier máquina que tenga instalada la **Java Runtime Environment (JRE)**, lo que facilita su despliegue y uso.

- Todos los frames tendran un boton llamado mostrara que permitira devolverse al frame anterior.
- En la mayoria de los frames encontrara try-catch para evitar excepciones.

Volver

Manejo de Bodega

Administra el inventario de productos del almacén, permitiendo agregar nuevos productos, modificar sus cantidades, registrar compras a proveedores, y retirar productos del inventario.

Manejo de Ventas

Controla todo el ciclo de ventas, desde la verificación del inventario, el registro de las ventas y la emisión de facturas, hasta el reporte diario de ventas. Además, el sistema garantiza que no se puedan realizar ventas si no hay suficiente inventario o si el saldo en caja es insuficiente

Devoluciones

Gestiona las devoluciones de productos, verificando que la caja cuente con suficiente dinero para realizar reembolsos. Los productos devueltos se descartan del inventario.

Cliente frecuente

Este módulo gestiona el programa de fidelización de clientes, que permite ofrecer descuentos a los clientes según su nivel de compras. Los niveles de clientes (Regular, VIP y Platino) son asignados en función del historial de compras.

Reporte Diario

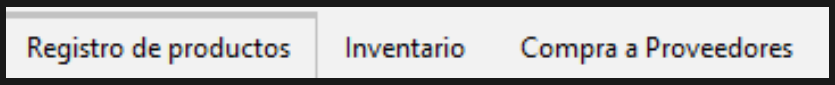
Administra el inventario de productos del almacén, permitiendo agregar nuevos productos, modificar sus cantidades, registrar compras a proveedores, y retirar productos del inventario.

MANEJO DE BODEGA

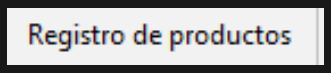
2

Administra el inventario de productos del almacén, permitiendo agregar nuevos productos, modificar sus cantidades, registrar compras a proveedores, y retirar productos del inventario. Además, mantiene un control sobre las cantidades mínimas de productos disponibles y ajusta los precios de venta en función del precio de compra.

Manejo de bodega cuanta con un Tabbed Pane el cual permite diferenciar todas las funciones que son necesarias para la utilizacion de la bodega e inventario.

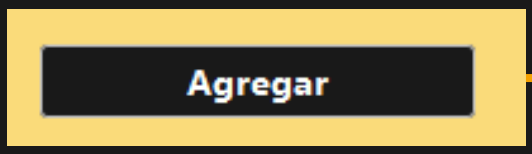


¿Cómo funciona?



La funcionalidad de Registro de Productos en el sistema se encarga de permitir al usuario ingresar nuevos productos al inventario del almacén.

Una vez que se completan los campos, el usuario presiona el botón "Agregar", que registra el producto en el sistema. Este registro valida que el producto sea agregado en la posición correcta dentro del arreglo bodega sin necesidad de un reordenamiento posterior. De esta manera, los productos estarán ordenados automáticamente por código, y el inventario se actualizará con la nueva información ingresada.



En la programación de este botón usamos Try-Catch para manejar posibles excepciones.

```
private void agregarActionPerformed(java.awt.event.ActionEvent evt) {
    try { //usamos Try-Catch para manejar posibles excepciones

        // Datos del nuevo producto
            int codigoProducto = Integer.parseInt(cod.getText());
            if (codigoProducto<1) {
              throw new NumberFormatException("El código debe ser un número positivo.");
```

Se hacen las validaciones pertinentes como se uestra en la imagen sumada a esta que todos los numeros que sean ingresados sean positivos, que la cantidad minima de el producto sea 5 y asegurandonos que tengamos suficiente espacio en bodega.

```
// Buscar la posición de inserción
    int pos= InsertarEn(codigoProducto);

// Desplazar elementos hacia la derecha (ajustando los índices)
    for (int i = N_Productos - 1; i >= pos; i--) {
        bodega[i + 1] = bodega[i];
    }

// Insertar el nuevo elemento
    bodega[pos] = X;
    N_Productos++;

inventario.actualizar(bodega, N_Productos);
mensaje.setText("Producto añadido con éxito");
```

Se busca la posicion del producto tomando de referencia el codigo.

Se realiza la sumatoria del producto y se inserta en su posicion.

Se relaciona la informacion de los productos con el boton actualizar y se le deja saber al usuario que la informacion se agrego correctamente.

MANEJO BODEGA

2

Inventario

Muestra el inventario actual en formato de tabla, también permite buscar productos por código, así mismo actualizar y eliminar información de productos.

Buscar

Si es que se encuentra el código en el sistema, se limpiara la tabla y solo se mostrara la información de ese producto.

```
private void buscarActionPerformed(java.awt.event.ActionEvent evt) {
    try{
    int codb= Integer.parseInt(codi.getText());
    if(Posicion(codb)==-1){
        throw new NumberFormatException("No existe el código en el sistema.");
    }

    limpiar(tabla);
        int pos= Posicion(Integer.parseInt(codi.getText()));
        // Mostrar los datos del código buscado
            Object[] datos = {bodega[pos][0], bodega[pos][1], bodega[pos][2],bodega[pos][3]};
            tabla.addRow(datos);
```

Compra a Proveedores

Facilita el registro de compras a proveedores, también actualiza las cantidades de productos existentes o añade nuevos productos, por último, se integra con el sistema de gestión de caja.

Luego de validar toda la informcaion si es que el saldo de la caja es suficiente se realiza la compra.

```
double egreso= cantidadProducto*costoProducto;
    if (egreso<Caja.getValorCaja()) {
        throw new NumberFormatException("La compra no puede ser realizada, no hay suficie

procesarProducto(codigoProducto, cantidadProducto, costoProducto, precioVent);
inventario.actualizar(bodega, N_Productos);
//Se actualiza caja
Caja.egresar(cantidadProducto*costoProducto);

mensaje2.setText("Compra realizada");
N_Compras++;
Total_Compras+=(cantidadProducto*costoProducto);
reporte.datosCompra(N_Compras, Total_Compras);
```

Se agrega la compra a inventario y se actualiza la caja.

Se notifica al usuario que la accion se realizo con exito, se agrega a la sumatoria del numero de compras y a la sumatoria del monto de esas compras para luego pasar esas sumatorias a reportes.

MANEJO DE VENTAS

3

Controla todo el ciclo de ventas, desde la verificación del inventario, el registro de las ventas y la emisión de facturas, hasta el reporte diario de ventas. Además, el sistema garantiza que no se puedan realizar ventas si no hay suficiente inventario o si el saldo en caja es insuficiente


```
int subtotal = (int) 0.0;
private int[] ag = new int[1000];
private int agCount = 0;
```

Se declaran antes de la accion del boton.

Agregar

```
try {
 // Obtener los valores
 int cod = Integer.parseInt(codigo.getText());
 if (cod < 1) {
     throw new NumberFormatException("El código debe ser un número positivo."
 }

 int pos = posicion(cod);
 if(pos == -1){
     throw new NumberFormatException("El código del producto no se encuentra
 }

 int cant = (int) cantidad.getValue();
 if (cant < 1) {
     throw new NumberFormatException("La cantidad debe ser mínimo 1.");
 }
```

Se realizan las validaciones pertinentes y nos aseguramos que el codigo del producto este realmente en bodega.

Agregar

```
int pre = bodega[pos][3];
int total = cant * pre;
```

Se identifica el precio y se calcula el total.

```
// Verificar que no se exceda el tamaño del arreglo
if (agCount + 4 > ag.length) {  // 4 espacios por producto
    JOptionPane.showMessageDialog(this, "Se ha alcanzado el li
    return;
}

// Agregar los valores al arreglo unidimensional
ag[agCount] = cod;    // Código
ag[agCount + 1] = cant;  // Cantidad
ag[agCount + 2] = pre;   // Precio
ag[agCount + 3] = total; // Total
agCount += 4;  // Incrementar en 4 porque son 4 datos por proc
```

Se verifica que no se exceda el tamaño, tambien se crean los 4 espacios para la informacion de los productos y se agrega la informacion al arreglo.

```
if(cant==bodega[pos][1] || bodega[pos][1]-cant<5){
    //Si la diferencia es 0 o es menor a 5, entonces eliminamo
    for (int i = pos; i < p - 1; i++) {
        bodega[i] = bodega[i + 1];
    }
    p--;
    inventario.actualizar(bodega,p);
} else{
    //De lo contrario, le restamos la cantidad vendida al stoc
    bodega[pos][1]-=cant;
}
```

Se verifica la cantidad del producto en bodega para ver si esta en circulacion si no se elimina, luego se actualiza inventario si esta en circulacion se le reduce la cantidad que se compro en bodega.

```
subtotal += total;
actualizarTextArea();
codigo.setText("");
cantidad.setValue(0);
```

Se calcula el total y se llama a el metodo “actualizarTextArea” la cual agrega la informacion al textarea y se limpian los textfield para poder agregar nueva informcaion.

```
private void actualizarTextArea() {
    String s = "Código\tCantidad\tTotal\n";// Encabezados de columnas

    for (int i = 0; i < agCount; i += 4) { // Itera de 4 en 4
        s += ag[i] + "\t" + ag[i + 1] + "\t" + ag[i + 3] + "\n";
        resultado.setText(s.toString());
    }
}

public int[] getProductos() {
return ag;
}
```

MANEJO DE VENTAS

3

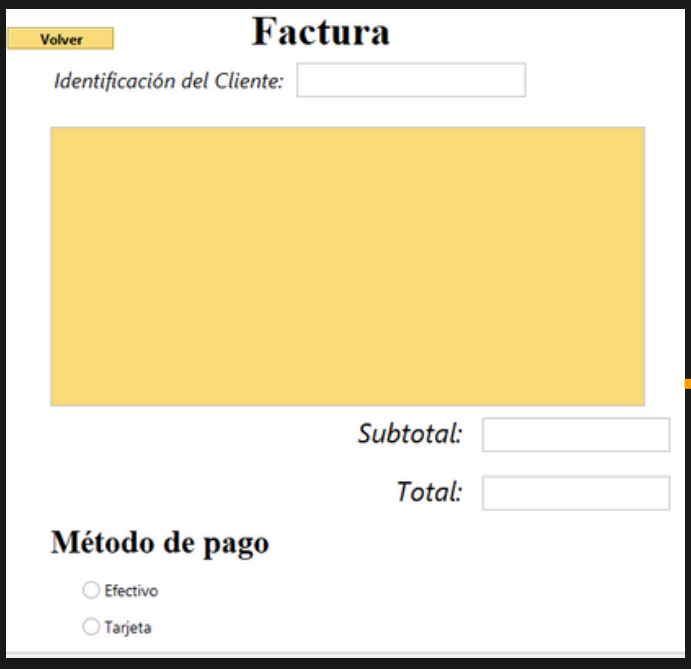
Finalizar compra

Se valida identificacion como numero positivo

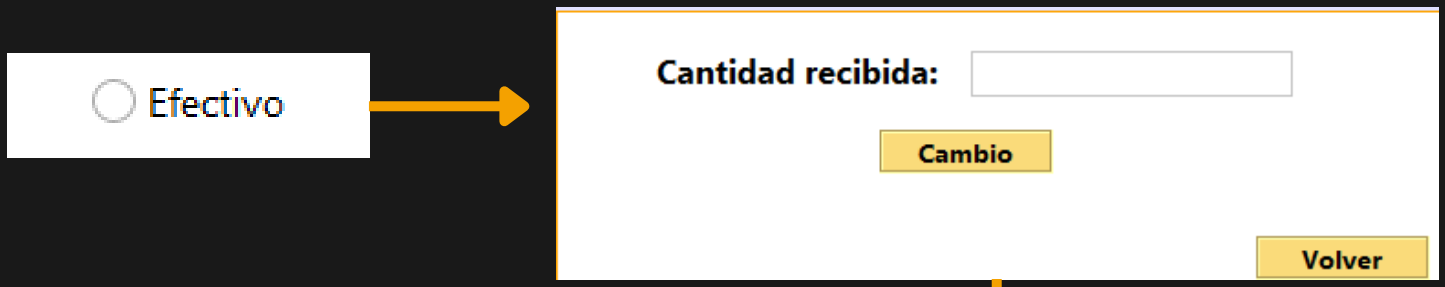
Se realiza las sumatorias de numero de ventas y del total del monto de venta y se pasan a reportes.

Se pasa identificacion, subtotal, total y la informacion de los productos a el frame factura.

```
private void finalizarActionPerformed(java.awt.event.ActionEven
    try{
        int id= Integer.parseInt(identificacion.getText());
        if (id < 1) {
            throw new NumberFormatException("La identificación deb
        }
        desc(id);

        N_Ventas++;
        Total_Ventas+=(subtotal*descuento);
        reporte.ventas(N_Ventas, Total_Ventas);

        //Se actualiza caja
        Caja.ingresar(subtotal*descuento);

        facturacion sv =new facturacion();
        sv.identi(Integer.parseInt(identificacion.getText()));
        sv.text( resultado.getText());
        sv.setSubtotal(subtotal);
        sv.setTotal(subtotal*descuento);
        sv.setVisible(true);

    } catch(NumberFormatException exc){

        JOptionPane.showMessageDialog(this, "Error: " + exc.get
    }
}
```



Se le realiza la factura al cliente y ayudamos al usuario a calcular el cambio.




```
private void CambioActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        int cantidadRecibida = Integer.parseInt(cantidad.getText());
        if (cantidadRecibida < 0) {
            JOptionPane.showMessageDialog(this, "La cantidad recibida no puede ser negativa.", "Er
        } else if (cantidadRecibida >= total) {
            double cambio = cantidadRecibida - total;
            JOptionPane.showMessageDialog(this, "Cambio: $" + cambio, "Cambio", JOptionPane.INFORM
        } else {
            JOptionPane.showMessageDialog(this, "La cantidad recibida debe ser mayor o igual al su
        }
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(this, "Por favor, ingrese un número válido para la cantidad
    }
}
```

Se valida la cantidad recibida que no debe ser negativa ni menor al total. luego se camcula el cambio que el cliente debe recibir.

DEVOLUCIONES

4

Controla todo el ciclo de ventas, desde la verificación del inventario, el registro de las ventas y la emisión de facturas, hasta el reporte diario de ventas. Además, el sistema garantiza que no se puedan realizar ventas si no hay suficiente inventario o si el saldo en caja es insuficiente.


```
private int[][] devoluciones = new int[100][2];


//Traer la información de Bodega
manejo_bodega.Inventario inventario = new Inventario();
private int [][] bodega = inventario.setBodega();
private int p= inventario.setProductos();


//Pasar la información de Compras al Reporte diario
private reporte.Actualizar reporte = new reporte.Actualizar();
int N_Devol = 0;
double Total_Devol=0;

Principal.Caja Caja = new Caja();
double disponible= Caja.getValorCaja();
```

Se traen al frma la informacion pertinente para el funcionamiento de las devoluciones como lo son bodega, inventario, caja. Tambien es necesario inicializar la matriz devoluciones y numero de devoluciones y total de devoluciones para pasarlo a reportes.

Realizar

```
int cod = Integer.parseInt(codigo.getText());
if (cod<1) {
        throw new NumberFormatException("El código debe ser un número positi
          }
    if(Posicion(cod)==-1){
        throw new NumberFormatException("La devolución no puede ser realizada,
    }
int cant= (int) cantidad.getValue();
    if (cant<1) {
        throw new NumberFormatException("La cantidad debe ser minimo 1.");
    }
```

Se realizan las validaciones pertinentes en este caso que todos los numeros sean positivos y nos aseguramos que el codigo se encuentre en bodega.

Se declara pos, precio y egreso ya que estos son claves para determinar si la devolucion puede ser hecha.

Se almacena la informacion en el vector "devoluciones".

Se pasan las sumatorias a reportes.

Se actualiza caja, se limpian los campos de texto y se le deja saber al usuario que la devolucion se ha hecho correctamente.

```
int pos= Posicion(cod);
int precio= bodega[pos][3];
double egreso= cant*precio;
    if (egreso>disponible) {
        throw new NumberFormatException("La devolución no
```

```
// Almacenar los datos en la matriz
devoluciones[N_Devol][0] = cod;
devoluciones[N_Devol][1] = cant;
```

```
N_Devol++;
Total_Devol+=cant*precio;
reporte.devol(N_Devol, Total_Devol);
```

```
//Se actualiza caja
Caja.egresar(cant*precio);

// Limpiar los campos de texto
codigo.setText("");
cantidad.setValue(0);

// Mostrar mensaje de confirmación
mensaje.setText("La devolución ha sido realizada");
```

Existe un metodo llamado posicion que busca la posicion del codigo dado en bodega y si es que este existe alli.

CLIENTE FRECUENTE

5

Este módulo gestiona el programa de fidelización de clientes, que permite ofrecer descuentos a los clientes según su nivel de compras. Los niveles de clientes (Regular, VIP y Platino) son asignados en función del historial de compras.


```java
public static class Clientes {
    private static int cl = 0;
    private static double[][] clientes = new double [100][5];

    public static synchronized void actualizar(double[][] CLIENTES , int CL)
        clientes=CLIENTES;
        cl= CL;
    }
    public static synchronized double[][] set() {
        return clientes;
    }
    public static synchronized int N(){
        return cl;
    }
}
```

Con esta clase se declara la matriz cliente que se pasara al resto de frames.

```java
boolean existe(int ID) {
    for (int j = 0; j < cl; j++) {
        if (ID == clientes[j][0]) {
            return true;
        }
    }
    return false; //
}
```

En este metodo nos aseguramos de que el ID relamente esta en la matriz, si no no podra pedir su descuento.

```java
int buscarPosicion(int ID) {
    for (int j = 0; j < cl; j++) {
        if ( ID< clientes[j][0]) {
            return j; // Encontramos la posición correcta
        }
    }
    return cl;  // Si no encontramos una posición, lo agregamos al final
}
```

Se organiza la matriz de manera que se encuentre la posicion o se coloque como ultimo.

```
amigo_frec.Clientes CLIENTES =new Clientes();
private double[][] clientes= CLIENTES.set();

@SuppressWarnings("unchecked")
Generated Code
int cl=0;
```

Se declara amigo_frec clientes y cl (contador de clientes).

Registrar

Se realizan las validaciones necesarias (que todos los numeros sean positivos).

```
try {
int ident= Integer.parseInt(id.getText());
if (ident<1) {
            throw new NumberFormatException("La identificación debe ser un número positivo.");
              }
if(existe(ident)){
    throw new NumberFormatException("Ya existe un cliente con el ID ingresado.");
        }

int ac= (int) acum.getValue();
if (ac<1) {
            throw new NumberFormatException("La cantidad acumulada en compras debe ser un núme
              }
```

```
int pos= buscarPosicion(ident);
clientes[pos][0]=ident;
clientes[pos][1]=ac;
```

Identificamos la pocision e insetamos la informacion pertirnrnte (la identificacion y el acumulado).

```
if(ac>500000){
    clientes[pos][2]= 0.7; //Cliente Platino 30%
} else{
    if(ac>100000){
        clientes[pos][2]= 0.8; //Cliente V.I.P 20%
    } else{
        clientes[pos][2]= 0.9; //Cliente Regular 10%
    }
}
```

Identificamos a que tipo de descuento pertenece con respecto a sus compras acumuladas (ac).

```
cl++;
```

Se realiza la sumatoria.

REPORTE DIARIO

6

Administra el inventario de productos del almacén, permitiendo agregar nuevos productos, modificar sus cantidades, registrar compras a proveedores, y retirar productos del inventario.


```java
public static class Actualizar {
    private static int N_Compras= 0;
    private static int Total_Compras= 0;
    private static int N_Ventas= 0;
    private static double Total_Ventas= 0;
    private static int N_Devol= 0;
    private static double Total_Devol= 0;
```

Esta clase nos sirve para traer inicializar las variables y actualizarlas a medida que se agrege informacion.

```java
public static synchronized void compras(int N, int TOTAL) {
    Total_Compras = TOTAL;
    N_Compras= N;
}
public static synchronized void ventas(int N, double TOTAL) {
    Total_Ventas = TOTAL;
    N_Ventas= N;
}
public static synchronized void devol(int N, double TOTAL) {
    Total_Devol = TOTAL;
    N_Devol= N;
}

//TRAER LA INFORMACIÓN
public static synchronized int totalc(){
    return Total_Compras;
}
public static synchronized int c(){
    return N_Compras;
}

public static synchronized double totalv(){
    return Total_Ventas;
}
public static synchronized int v(){
    return N_Ventas;
}
```

Se actualiza y se trae la informacion que se estuvo recolectando en todos los frames.

```
reporte.Actualizar Actualizar = new Actualizar();
    private final int N_Compras= Actualizar.c();
    private final int Total_Compras= Actualizar.totalc();

    private final int N_Ventas= Actualizar.v();
    private final double Total_Ventas= Actualizar.totalv();

    private final int N_Devol= Actualizar.d();
    private final double Total_Devol= Actualizar.totald();

Principal.Caja Caja= new Caja();
```

Se declaran las variables y se relacionan con actualiza, tambien declaramos caja.

Mostrar

```
private void mostrarActionPerformed(java.awt.event.ActionEvent evt) {
    Ventas.setText(Integer.toString(N_Ventas));
    TotalVentas.setText(Double.toString(Total_Ventas));

    Devoluciones.setText(Integer.toString(N_Devol));
    TotalDevoluciones.setText(Double.toString(Total_Devol));

    Compras.setText(Integer.toString(N_Compras));
    TotalCompras.setText(Integer.toString(Total_Compras));

    totalcaja.setText(Double.toString(Caja.getValorCaja()));
}
```

Se muestran las variables en los campos de texto correspondientes.

CONTACTO

Si tienes preguntas sobre el funcionamiento del software o requieres asistencia técnica, puedes ponerte en contacto con las creadoras del sistema o el equipo de soporte de Almacenes de La Prosperidad a través de los siguientes medios:

Creadoras del Software

Este software fue desarrollado por:

Katherin Barrera y Valeria Florez.

Correo Electrónico de las Creadoras:

Para consultas técnicas o información sobre el desarrollo, puedes escribir a:

- lkatherin@uninorte.edu.co
- florezvaleria@uninorte.edu.co