Manual Técnico - Sistema de Gestión de Inventario

Descripción General

Sistema de gestión de inventario desarrollado en Java que permite administrar productos, registrar ventas, generar reportes en PDF y mantener un registro completo de bitácora de todas las operaciones realizadas.

Requerimientos del Sistema

Requerimientos Funcionales

- Gestión de productos (crear, buscar, eliminar)
- Control de inventario y stock
- Registro de ventas con validación de stock
- Generación de reportes en PDF
- Sistema de bitácora para auditoría
- Interfaz de consola intuitiva

Requerimientos No Funcionales

- Java JDK 8 o superior
- Biblioteca iTextPDF v5.5.13.3
- 2 GB de RAM mínimo
- 100 MB de espacio en disco
- Sistema operativo: Windows, Linux o macOS

Estructura de Clases

Clase: producto.java

Representa un producto en el inventario.

```java
public class producto {
    // Clase Producto
// Representa un producto de la tienda (camisa, pantalón, etc.)

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

    // Constructor
    public producto(String codigo, String nombre, String categoria, double precio, int cantidad) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.categoria = categoria;
        this.precio = precio;
        this.cantidad = cantidad;
    }

    // Método para mostrar la información del producto
    public void mostrarInfo() {
        System.out.println(codigo + " | " + nombre + " | " + categoria +
                           " | Q" + precio + " | Stock: " + cantidad);
    }
}
```

Atributos:

- `codigo`: Identificador único del producto
- `nombre`: Nombre del producto
- `categoria`: Categoría del producto
- `precio`: Precio unitario
- `cantidad`: Stock disponible

Métodos:

- `producto(String codigo, String nombre, String categoria, double precio, int cantidad)`
    - Constructor que inicializa todos los atributos
- `void mostrarInfo()`
    - Muestra la información del producto en consola

Clase: inventario.java

```
    import java.io.FileWriter;
/**
 *
 *
 */
public class inventario {


// Clase Inventario
// Maneja las operaciones sobre el vector de productos

    producto[] productos = new producto[100]; // vector de productos
    int contador = 0;

    // Agregar producto validando código único
    public boolean agregarProducto(producto p) {
        for (int i = 0; i < contador; i++) {
            if (productos[i].codigo.equals(p.codigo)) {
                return false; // código repetido
            }
        }
        productos[contador] = p;
        contador++;
        return true;
    }
```

Gestiona el conjunto de productos y operaciones relacionadas.

Atributos:

- `productos[]`: Array de productos (capacidad: 100)
- `contador`: Número actual de productos

Métodos:

- `boolean agregarProducto(producto p)`
  - Agrega un producto al inventario verificando código único
  - Retorna true si fue exitoso, false si el código ya existe
- `producto buscarPorCodigo(String codigo)`
  - Busca un producto por su código
  - Retorna el producto o null si no existe
- `boolean eliminarProducto(String codigo)`
  - Elimina un producto del inventario

    - Retorna true si fue exitoso, false si el producto no existe
- `void mostrarProductos()`
    - Muestra todos los productos en el inventario
- `boolean registrarVenta(String codigo, int cantidad)`
    - Registra una venta, validando stock disponible
    - Actualiza el stock y guarda la venta en archivo
    - Retorna true si fue exitoso, false si hay error

Clase: bitacora.java

```java
import java.time.LocalDateTime;

public class bitacora {
    String fechaHora;
    String accion;
    boolean correcta;
    String usuario;

    public bitacora(String accion, boolean correcta, String usuario) {
        this.fechaHora = LocalDateTime.now().toString(); // guarda fecha y hora actual
        this.accion = accion;
        this.correcta = correcta;
        this.usuario = usuario;
    }

    public void mostrarInfo() {
        System.out.println("[" + fechaHora + "] Usuario: " + usuario +
                           " | Acción: " + accion +
                           " | Resultado: " + (correcta ? "Correcta" : "Errónea"));
    }
}
```

Representa un registro individual en la bitácora del sistema.

Atributos:

- `fechaHora`: Timestamp de la operación
- `accion`: Descripción de la acción realizada
- `correcta`: Estado de éxito/error de la operación
- `usuario`: Usuario que realizó la operación

Métodos:

- `bitacora(String accion, boolean correcta, String usuario)`
  - Constructor que inicializa el registro con timestamp automático
- `void mostrarInfo()`
  - Muestra la información del registro de bitácora

Clase: gestorBitacora.java

```java
public class gestorBitacora {

    bitacora[] registros = new bitacora[200]; // máximo 200 acciones
    int contador = 0;

    // Agregar un registro
    public void registrarAccion(String accion, boolean correcta, String usuario) {
        registros[contador] = new bitacora(accion, correcta, usuario);
        contador++;
    }

    // Mostrar todas las acciones
    public void mostrarBitacora() {
        if (contador == 0) {
            System.out.println("No hay acciones registradas.");
        } else {
            for (int i = 0; i < contador; i++) {
                registros[i].mostrarInfo();
            }
        }
    }
}
```

Gestiona el conjunto de registros de bitácora.

Atributos:

- `registros[]`: Array de registros (capacidad: 200)
- `contador`: Número actual de registros

Métodos:

- `void registrarAccion(String accion, boolean correcta, String usuario)`

    - Crea y almacena un nuevo registro en la bitácora
- `void mostrarBitacora()`
    - Muestra todos los registros de la bitácora

Clase: venta.java

```java
public class venta {
    // Clase Venta
// Guarda la información de una transacción realizada
    String codigoProducto;
    int cantidadVendida;
    String fechaHora;
    double total;

    public venta(String codigoProducto, int cantidadVendida, String fechaHora, double total) {
        this.codigoProducto = codigoProducto;
        this.cantidadVendida = cantidadVendida;
        this.fechaHora = fechaHora;
        this.total = total;
    }

    public void mostrarInfo() {
        System.out.println("Producto: " + codigoProducto + " | Cantidad: " + cantidadVendida +
                           " | Total: Q" + total + " | Fecha: " + fechaHora);
    }
}
```

Representa una transacción de venta.

Atributos:

- `codigoProducto`: Código del producto vendido
- `cantidadVendida`: Cantidad vendida
- `fechaHora`: Timestamp de la venta
- `total`: Monto total de la venta

Métodos:

- `venta(String codigoProducto, int cantidadVendida, String fechaHora, double total)`
    - Constructor que inicializa la venta
- `void mostrarInfo()`
    - Muestra la información de la venta

Clase: reportePdf.java

```java
  import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.PdfPTable;
import com.itextpdf.text.pdf.PdfWriter;
import java.io.FileOutputStream;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

/**
 *
 * @author ramirito
 */
public class reportePdf {



    // Método para generar nombre del archivo con fecha y hora
    private static String generarNombre(String tipo) {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd_MM_yyyy_HH_mm_ss");
        String fecha = LocalDateTime.now().format(dtf);
        return fecha + "_" + tipo + ".pdf";
    }
```

Genera reportes en formato PDF.

Métodos:

- `static String generarNombre(String tipo)`
  - Genera nombre único para archivos PDF con timestamp
- `static void generarReporteStock(inventario inventario)`
  - Genera reporte PDF con el inventario actual
- `static void generarReporteVentas(String archivoVentas)`
  - Genera reporte PDF con el historial de ventas

Clase: abrirPdf.java

```java
import java.awt.Desktop;
import java.io.File;

public class abrirPdf {
    public static void abrir(String ruta) {
        try {
            File file = new File(ruta);
            if (file.exists()) {
                Desktop.getDesktop().open(file);
            } else {
                System.out.println("El archivo no existe: " + ruta);
            }
        } catch (Exception e) {
            System.out.println("Error al abrir PDF: " + e.getMessage());
        }
    }
}
```

Utilidad para abrir archivos PDF.

Métodos:

- `static void abrir(String ruta)`
    - Abre el archivo PDF con la aplicación predeterminada

Clase: main.java

```java
import java.util.Scanner;

public class main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        inventario inventario = new inventario();
        gestorBitacora bitacora = new gestorBitacora();

        String usuario = "Estudiante"; // se puede personalizar
        int opcion;

        do {
            System.out.println("\n--- Sistema de Inventario ---");
            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. Ver Inventario");
            System.out.println("6. Ver datos del estudiante");
            System.out.println("7. Ver Bitácora");
            System.out.println("8. Generar reporte de stock");
            System.out.println("9. Generar reporte de inventario");
            System.out.print("Opción: ");
            opcion = sc.nextInt();
            sc.nextLine();
```

Clase principal con el menú interactivo.

Métodos:

- `static void main(String[] args)`
    - Punto de entrada principal del programa
    - Implementa el menú interactivo y flujo de control

Diagrama de Clases Simplificado

```
[main] → [inventario] → [producto]
        → [gestorBitacora] → [bitacora]
        → [reportePdf] → [abrirPdf]

        → [venta]
```

Flujo de Datos

1. Usuario interactúa con el menú en `main.java`
2. Las operaciones se delegan a `inventario.java`
3. Cada operación se registra en `gestorBitacora.java`
4. Las ventas se guardan en archivo y en `venta.java`
5. Los reportes se generan con `reportePdf.java`

Validaciones Implementadas

- Unicidad de código de producto
- Campos no vacíos (nombre, categoría)
- Valores numéricos válidos (precio > 0, cantidad ≥ 0)
- Stock suficiente para ventas
- Manejo de excepciones en entrada de datos

Formatos de Almacenamiento

- Memoria: Arrays de objetos para productos y bitácora
- Persistencia: Archivo de texto "ventas.txt" para historial de ventas
- Reportes: Archivos PDF con nombre timestamped

Limitaciones Técnicas

- Capacidad fija: 100 productos, 200 registros de bitácora
- No persistencia completa entre ejecuciones
- Dependencia externa de iTextPDF

Consideraciones de Rendimiento

- Búsqueda lineal O(n) en arrays
- Operaciones básicas con complejidad constante O(1)
- Adecuado para volúmenes pequeños de datos

Dependencias Externas

- iTextPDF v5.5.13.3: Para generación de reportes PDF
- Java AWT Desktop: Para apertura de archivos PDF

Mensajes de Error

El sistema incluye mensajes de error descriptivos para:

- Códigos de producto duplicados
- Stock insuficiente
- Entradas inválidas
- Errores de archivo
- Problemas con PDF

Métodos de Depuración

- Bitácora detallada de todas las operaciones
- Mensajes de consola informativos
- Validación exhaustiva de entradas