Xhunik Nikol Miguel Mutzutz

Carnet: 201900462

Introducción a la computación y programación 1
Manual Tecnico

Fecha: 22 de diciembre del 2019

Índice

Introducción

Objetivos y alcances del manual

El presente manual técnico presenta, a grandes rasgos, la totalidad de los detalles técnicos de la aplicación implementada, detallando el funcionamiento interno del programa, dividiéndolo en 2 secciones principales, la parte lógica del programa y la parte de visualización, cada una de la cuales, son gestionadas independientemente, y por lo tanto son descritas por separado.

El manual pretende describir el funcionamiento interno de las clases definidas, así como las instancias de las mismas creadas durante la ejecución, las cuales son utilizadas para poder controlar el flujo del programa en un entorno dinámico, donde no se conoce a profundidad las condiciones iniciales del programa.

Objetivos y alcances de la aplicación

La aplicación para control de Inventario pretende ser de utilidad en cualquier ámbito, de manera que la aplicación de manera transparente al operador maneja el control de insumos disponibles para la fabricación de los productos, de tal manera que controla que todas las operaciones sean validadas antes de ser autorizadas, asegurando así la consistencia entre los datos manejados y el estado real de los inventarios utilizados.

Esta aplicación está preparada, para funcionar con cualquier cantidad de insumos y de productos sin necesidad de readaptar los componentes, además de proveer herramientas para poder realizar todas las operaciones de manera cómoda, al contar con funciones que proveen características como búsqueda y ordenamiento.

Descripción General de la Aplicación

La aplicación consta de 5 módulos en la parte gráfica, los cuales son los que interactúa el usuario, en la parte lógica del programa la cual se encarga de controlar el flujo de datos, así como de realizar las validaciones, mantener cargados los datos que se requieran, realizar las búsquedas y leer los archivos de inventario en formato XML desde disco.

Parte Gráfica

La parte gráfica se encuentra programada con la librería Swing, la parte gráfica se dividió en varios módulos los cuales proporcionan una característica diferente, para proporcionar las funcionalidades, la parte gráfica llama a los métodos de los objetos manejado por la parte Lógica, los cuales se encargan de modificar y devolver los datos para que puedan ser utilizados por la parte gráfica.

Parte Lógica

La parte lógica del programa consta de 2 elementos principales, la Clase Producto y la Clase Insumo, además de 2 Clases auxiliares diseñadas para poder manejar un cantidad variables de objetos, la Clase ListaProducto y la Clase ListaInsumo, el control de dinero, insumos disponibles y las ventas son gestionadas por objetos de la Clase Inventario, Gestor y PrincipalControl, las cuales implementan toda las validaciones para realizar una venta, así como las acciones a realizar durante la autorización de la venta.

Características Técnicas

La aplicación se desarrolló utilizando:

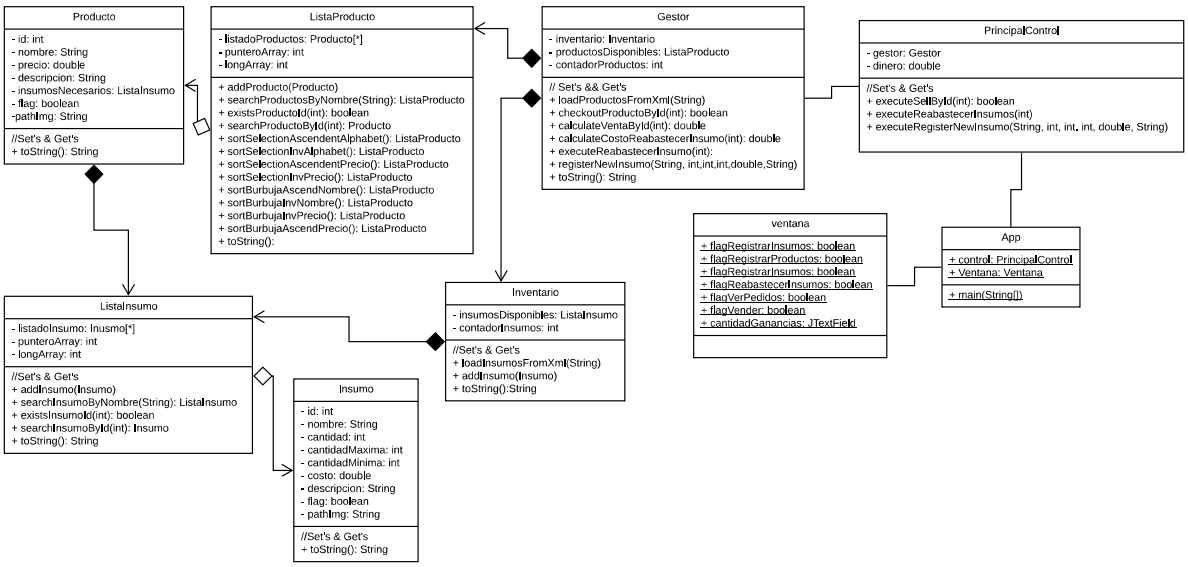
- Entorno de Desarrollo Integrado: Apache NetBeans 11.0 (Apache Foundation)
- Lenguaje utilizado: Java (SunMicrosystems -> Oracle)
- Versión de java: OpenJdk 8.0 (SunMicrosystems - > RedHat)
- Librería Gráfica: javax.Swing y java.awt
- Lectura XML: org.w3c dom 2.3.0-jaxb-1.0.6
- Herramienta de Control y Gestión de Proyecto: Apache Maven (Apache Foundation)

Diseño de la aplicación

Diagrama de Clases

Descripción del diseño

Clase Producto y Clase Insumo



Se crearon las dos clases que contienen los elementos que debe manejar la aplicación, cada una de ellas tiene las propiedades necesarias para poder operar.

Como clases auxiliares se crearon las Clases ListaProducto y LIstaInsumo, las cuales implementan métodos para añadir nuevas entradas de Producto e Insumo, redimensionando los arreglos que manejan internamente de tal manera que virtualmente no existe ningún límite a la cantidad de Ítems que pueden ser cargados, y por tanto utilizados dentro de la aplicación.

Clases Lista

Estas clases son instanciadas una única vez por el Gestor y el Inventario, las cuales dentro de sí implementan los métodos para ordenar y buscar según los criterios enviados a través de PrincipalControl desde la interfaz, estas dos clases funcionan como una implementación de memoria Dinámica al simular el funcionamiento de la misma redimensionando el array que contiene los elementos cada vez que sea necesario.

Clase Inventario

Esta clase contiene dentro de sí una instancia de ListaInsumo, en la cual lleva el control de todos los insumos que están disponibles, así como los valores máximos y mínimos que pueden encontrarse en existencia, dentro de esta clase se

encuentra la función que carga la lista de insumos desde el disco, la cual se encarga de añadir a la lista de insumos todos los elementos para poder ser consultados posteriormente.

Clase Gestor

Esta clase contiene una instancia de inventario, y una de ListaProductos, esta clase es la que se encarga de controlar si los insumos existentes son suficientes para la fabricación de un producto, esto es debido a que no existen como tal una cantidad de productos, si no que estos están ligados a la disponibilidad de los insumos requeridos para su fabricación, y por lo tanto deben ser manejados al mismo tiempo.

Dentro de esta clase se implementan funciones para obtener el cálculo del dinero requerido para el reabastecimiento de los insumos, así como la implementación del método encargado de validar las ventas y el dinero obtenido de las Mismas.

Clase PrincipalControl

Esta clase es instanciada una única vez por la función main de manera estática, esto debido a que por ser la función main una función estática, las variables deben estar declaradas como tales, además de que de esta manera todos los objetos de la parte Gráfica pueden acceder a la misma a través de la clase App.

Esta clase provee el control del flujo del dinero al leer el resultado de las autorizaciones realizadas por la clase gestor, obteniendo la cantidad positiva o negativa de dinero que existe actualmente en caja.

Clase App

Esta clase contiene la función main y es la que provee el hilo principal de ejecución de la aplicación, en esta se encuentra una instancia de PrincipalControl y una instancia de Ventana, la cual hereda de la clase JFrame y es desde la cual se gestionan todos los eventos que se utilizan para realizar solicitudes al objeto de controlo definido en la clase App e inicializado en la función main.

Métodos, funciones y procedimientos

Para la implementación de las funcionalidades de la aplicación se utilizaron diversos algoritmos para ejecutar las instrucciones requeridas por la lógica interna del programa.

Ordenamiento Burbuja

Este algoritmo de ordenamiento se basa en realizar el ordenamiento, comparando dos elementos de la lista e intercambiandolos si estos se encuentran en desorden, para el funcionamiento del programa se utilizó este ordenamiento tanto en sentido ascendente como descendente, para la implementación se usaron los siguientes algoritmos:

```
//Burbuja Ascendente por nombre
public ListaProducto sortBurbujaAscendNombre(){
      Producto[] p = getListadoProductos();
      ListaProducto lista = new ListaProducto(1);
      for (int i = 0; i < getSize(); i++){
         for (int j = 0; j < getSize()-1;j++){
            if (p[j].getNombre().compareTo(p[j+1].getNombre())> 0) {
               Producto tmp = p[j];
               p[j] = p[j+1];
               p[j+1] = tmp;

            }
         }
      }
      for (int i = 0; i < p.length;i++){
         lista.addProducto(p[i]);
      }
      return lista;
   }
//Burbuja descendente por nombre
public ListaProducto sortBurbujaInvNombre(){
```

```
        Producto[] p = getListadoProductos();
        ListaProducto lista = new ListaProducto(1);
        for (int i = 0; i < getSize(); i++){
            for (int j = 0; j < getSize()-1;j++){
                if (p[j].getNombre().compareTo(p[j+1].getNombre())< 0) {
                    Producto tmp = p[j+1];
                    p[j+1] = p[j];
                    p[j] = tmp;

                }
            }
        }
        for (int i = 0; i < p.length;i++){
            lista.addProducto(p[i]);
        }
        return lista;
    }
//burbuja descendente por precio
public ListaProducto sortBurbujaInvPrecio(){
        Producto[] p = getListadoProductos();
        ListaProducto lista = new ListaProducto(1);
        for (int i = 0; i < getSize(); i++){
            for (int j = 0; j < getSize()-1;j++){
                if (p[j].getPrecio() < p[j+1].getPrecio()) {
                    Producto tmp = p[j+1];
                    p[j+1] = p[j];
                    p[j] = tmp;

                }
            }
        }
        for (int i = 0; i < p.length;i++){
            lista.addProducto(p[i]);
        }
        return lista;
    }
```

```
//burbuja ascendente por precio
    public ListaProducto sortBurbujaAscendPrecio(){
        Producto[] p = getListadoProductos();
        ListaProducto lista = new ListaProducto(1);
        for (int i = 0; i < getSize(); i++){
            for (int j = 0; j < getSize()-1;j++){
                if (p[j].getPrecio() > p[j+1].getPrecio()) {
                    Producto tmp = p[j];
                    p[j] = p[j+1];
                    p[j+1] = tmp;
                }
            }
        }
        for (int i = 0; i < p.length;i++){
            lista.addProducto(p[i]);
        }
        return lista;
    }
```

Los algoritmos implementados para comparar el nombre, utilizan la función de la clase String llamada compareTo, la cual devuelve la distancia entre las primeras dos letras distintas que encuentre entre ambas cadenas de texto.

Para obtener los resultados inversos, únicamente se cambió el sentido de la comparación y el orden de asignación de las variables para obtener el resultado en el sentido Opuesto.

Ordenamiento por selección

El funcionamiento del ordenamiento por selección consiste en explorar toda la lista hasta encontrar el mínimo, colocándolo en la primera posición, para posteriormente volver a realizar el mismo proceso desde la siguiente posición a la última evaluada, de manera que desde el momento en que son colocadas en su posición quedan en el orden correcto.

Las implementaciones para esta aplicación fueron:

```
//Ordenamiento por selección en orden alfabético ascendente
public ListaProducto sortSelectionAscendentAlphabet(){
        int n;
        ListaProducto lista = new ListaProducto(1);
        Producto[] p = getListadoProductos();

        for (int i = 1; i < getSize(); i++ ){
            Producto aux = p[i];
            n = i;
            while (n > 0 && p[n-1].getNombre().compareTo(aux.getNombre())>0){
                p[n] = p[n-1];
                --n;
            }
            p[n] = aux;
        }
        for (int i = 0; i < p.length;i++ ){
            lista.addProducto(p[i]);
        }
        return lista;
    }
```

```
//Ordenamiento por selección en orden alfabético descendente
    public ListaProducto sortSelectionInvAlphabet(){
        int n;
        ListaProducto lista = new ListaProducto(1);
        Producto[] p = getListadoProductos();

        for (int i = 1; i < getSize(); i++ ){
            Producto aux = p[i];
            n = i;
            while (n > 0 && p[n-1].getNombre().compareTo(aux.getNombre())<0){
                p[n] = p[n-1];
                --n;
            }
            p[n] = aux;
        }
        for (int i = 0; i < p.length;i++ ){
            lista.addProducto(p[i]);
        }
        return lista;
    }
    //Ordenamiento por selección en orden de precios ascendente
    public ListaProducto sortSelectionAscendentPrecio(){
        int n;
        ListaProducto lista = new ListaProducto(1);
        Producto[] p = getListadoProductos();

        for (int i = 1; i < getSize(); i++ ){
            Producto aux = p[i];
            n = i;
            while (n > 0 && p[n-1].getPrecio() > aux.getPrecio()){
                p[n] = p[n-1];
                --n;
            }
            p[n] = aux;
        }
        for (int i = 0; i < p.length;i++ ){
            lista.addProducto(p[i]);
        }
        return lista;
    }
```

```
//Ordenamiento por selección en orden de precios descendente
public ListaProducto sortSelectionInvPrecio(){
    int n;
    ListaProducto lista = new ListaProducto(1);
    Producto[] p = getListadoProductos();

    for (int i = 1; i < getSize(); i++ ){
        Producto aux = p[i];
        n = i;
        while (n > 0 && p[n-1].getPrecio() < aux.getPrecio()){
            p[n] = p[n-1];
            --n;
        }
        p[n] = aux;
    }
    for (int i = 0; i < p.length;i++ ){
        lista.addProducto(p[i]);
    }
    return lista;
}
```

Algoritmos de búsqueda

Para la implementación del programa, se utilizaron algoritmos de búsqueda para encontrar coincidencias en el nombre del producto y mostrar únicamente aquellos en los cuales existan coincidencias, por otro lado también se implementaron algoritmos de búsquedas, que se basan en encontrar el producto que coincide con una id única, esto para poder vincular los eventos de botón, independientemente de la posición que ocupen dentro del arreglo de productos e insumos disponibles.

```
public ListaProducto searchProductosByNombre(String nombre){
    ListaProducto listaTemporal = new ListaProducto(this.getSize());
    ListaProducto listaRetorno;
    int contadorCoincidencias = 0;
    for (int i = 0; i < this.getSize();i++){
        String[] palabras = nombre.split("\\s+");
        for (String palabra : palabras) {
            if (listadoProductos[i].getNombre().contains(palabra)) {
                System.out.println("Encontrado");
                listaTemporal.addProducto(listadoProductos[i]);
                contadorCoincidencias++;
            }
        }
    }
    listaRetorno = new ListaProducto(contadorCoincidencias);
    for (int i = 0; i < contadorCoincidencias;i++ ){
        if (listaTemporal.getProductoByIndex(i).isFlag()){
            listaRetorno.addProducto(listaTemporal.getProductoByIndex(i));
        }
    }
    return listaRetorno;
}
```

Este algoritmo hace uso de la función split dentro de la clase String, la cual separa la cadena en palabras independientes cada vez que encuentra el carácter de “espacio”, de tal manera que puede encontrar coincidencias incluso en nombres de múltiples palabras, independientemente de su posición, para encontrar el caracter espacio hace uso de la expresión regular “\\s+”, una vez encontradas las coincidencias, itera sobre las coincidencias para eliminar las posiciones vacías dentro de la lista de coincidencias encontradas.

Buscar Producto e insumo por número de Id

Se implementó un algoritmo que devuelve un objeto producto que contenga una id igual a la ingresada como argumento de búsqueda, este algoritmo únicamente devuelve un objeto Producto y no una lista de Productos, debido a que una id sólo puede estar asociada a un producto en particular.

```
public Producto searchProductoByID(int id){
      Producto productoRetorno = new Producto();
      for (int i = 0; i < this.getSize();i++){
          if (listadoProductos[i].getId() == id){
              productoRetorno = listadoProductos[i];
          }
      }
      return productoRetorno;
  }
```

El funcionamiento del algoritmo se basa en explorar toda la cadena hasta que encuentre una coincidencia, si no encuentra ninguno devuelve un objeto Producto con sus valores inicializados a 0.

```
public boolean existsProductoId(int id){
        for (int i = 0; i < this.getSize();i++){
            if (listadoProductos[i].getId() == id){
                return true;
            }
        }
        return false;
    }
```

Para evitar problemas, se implementó una función que verifica si existe un producto asociado a una id, devolviendo true si existe tal producto y false sino.

Glosario

- **Clase:** Plantilla de objetos
- **Objeto:** Instancia de una clase
- **Arreglo:** Conjunto de variables agrupadas bajo un mismo identificador
- **Algoritmo**: conjunto de instrucciones para realizar una acción, definido y finito.