**INFORME DE DESARROLLO**

**GESTIÓN DE INVENTARIO CON ARRAYLIST Y OBJETOS**

Jesús David Pérez Duran

Sebastián León Carrascal

Andrés Felipe Polo Manosalva

Programación Grupo C

Jesús Eduardo

Fecha de entrega: 02/05/2025

Universidad Francisco de paula de Santander

Norte de Santander

Ocaña

2025

INTROCUCCIÓN

En este informe se presenta el proceso y análisis para el desarrollo de una aplicación en Java, encargada de gestionar el inventario de productos de una tienda, mediante la implementación de ArrayList y métodos específicos, que permiten tener un control adecuado de los datos manejados. Esta herramienta permite almacenar diferentes datos del mismo tipo, actualizar, y calcular información sobre los productos, donde su capacidad de ser dinámica o flexible en cuanto a su tamaño tiene una gran ventaja frente a los arrays comunes, lo que nos permitió simular una gestión de inventarios dinámicos, facilitando así la inclusión de nuevos productos, la actualización de cantidades y el cálculo de los valores totales.

DESARROLLO DEL SISTEMA

**Paso 1. solicitud inicial de los productos**

Lo primero que se requiere es que el programa solicite los datos de, al menos, cinco productos y que pueda almacenar dicha información. Para la entrada de datos, se utilizó la clase Scanner, ya que es una forma eficiente de leer lo que el usuario ingresa y, además, permite la entrada de datos por consola, lo cual era uno de los requisitos a cumplir.

En cuanto a cómo almacenaríamos los datos, la mejor opción fue utilizar ArrayList , dado que inicialmente se pensó en emplear arrays convencionales, donde cada uno se encargaría de almacenar un tipo de dato por producto: uno para los nombres, otro para las cantidades y otro para los precios. Sin embargo, sabíamos que el programa debía permitir agregar nuevos productos y actualizar la información existente, lo cual implicaba que la estructura de datos debía poder expandirse dinámicamente. Como los arrays comunes no permiten esta flexibilidad, optamos por utilizar ArrayList, que sí cuenta con esa capacidad y, además, permite trabajar con datos de tipo objeto. Esta estructura nos ofrece una serie de métodos que facilitan una mejor gestión de los datos almacenados.

Teniendo esto en cuenta, se crearon tres ArrayList para guardar los nombres, las cantidades y los precios de los productos.

Finalmente, para garantizar que se ingresaran al menos cinco productos, se decidió preguntarle al usuario cuántos productos deseaba registrar, aclarando que el mínimo debía ser cinco. Mediante una condición en el código, se verificó que tanto el usuario como el programa respetaran esta cantidad mínima requerida. Y se utilizó un ciclo for que llegaba hasta la cantidad de productos que el usuario quería ingresar.

**Paso 2. permitir agregar más productos**

El programa debía permitir agregar más productos después de haber ingresado los cinco iniciales. Para lograr esto, se decidió preguntarle al usuario si deseaba realizar dicha acción. Como la respuesta podía ser "sí" o "no", era necesario evaluarla mediante una condición. Sin embargo, al tratarse de una cadena de texto (String), primero debíamos asegurarnos de que la respuesta estuviera en un formato específico: en minúscula y sin espacios adicionales.

Investigando qué métodos podían ayudarnos a normalizar la entrada, encontramos que trim().toLowerCase() era una combinación efectiva. El método trim() elimina los espacios al inicio y al final de la cadena, mientras que toLowerCase() convierte todo el texto a minúsculas. De este modo, garantizamos que la comparación con la palabra "sí" fuera precisa.

Este paso era crucial, ya que para evaluar la respuesta del usuario utilizamos el método .equals(), comparando su entrada con la cadena "sí". Si la condición se cumplía, es decir, si el usuario ingresaba "sí", entonces se procedía a permitir el ingreso de nuevos productos. Si, por el contrario, la respuesta era "no", el programa continuaba con la ejecución del resto del código.

En caso de que el usuario decidiera agregar más productos, era necesario implementar una lógica que permitiera capturar estos nuevos datos. Para ello, se utilizó una estructura de control do-while, la cual nos aseguraba que, al menos, se permitiera el ingreso de un producto adicional. Todos los datos ingresados eran leídos mediante la clase Scanner, y se agregaban a los respectivos ArrayList utilizando el método .add(), que permite añadir elementos a estas listas dinámicas.

Luego, dentro del mismo bucle do-while, se volvió a preguntar al usuario si deseaba agregar otro producto adicional al que acababa de ingresar. Esta lógica permitía que el ciclo continuara hasta que el usuario respondiera "no" o cualquier otra cosa diferente a "sí", lo cual detenía el ingreso de nuevos productos.

Cabe resaltar que no se le preguntó al usuario desde un principio cuántos productos adicionales deseaba agregar, ya que esta estrategia no resultaba óptima. Obligar al usuario a contar los productos nuevos que ha recibido, además de llevar un control sobre los ya ingresados, podía generar confusión o errores. Por esta razón, consideramos que nuestra solución era más eficiente y amigable para el usuario

**Paso 3. actualización de cantidades**

Además de permitir agregar nuevos productos, el programa también debía ofrecer la posibilidad de que el usuario pudiera modificar la cantidad de un producto ya existente en el inventario. Para determinar si el usuario deseaba realizar esta acción, se aplicó el mismo procedimiento utilizado en el paso anterior: se le preguntó al usuario y se evaluó su respuesta utilizando los métodos trim().toLowerCase() para normalizarla, y luego equals() para compararla con la palabra "sí".

En caso de que la respuesta fuera afirmativa, debíamos considerar varios aspectos importantes. Primero, para permitir la actualización de las cantidades de más de un producto, era necesario utilizar un bucle. Se optó nuevamente por la estructura do-while, la cual permite ejecutar al menos una vez el bloque de instrucciones. Dentro del ciclo, se volvió a preguntar si el usuario deseaba modificar la cantidad de otro producto. Según la respuesta, se podía detener o continuar el ciclo, permitiendo así múltiples modificaciones.

En cuanto a la lógica para actualizar la cantidad de un producto, llegamos a la conclusión de que lo primero era conocer el nombre del producto que se quería modificar. Esto era fundamental, ya que nos permitiría recorrer el ArrayList de nombres mediante un ciclo for para buscar la posición exacta en la que se encontraba dicho producto. Esa posición es esencial para poder modificar el valor correspondiente en el ArrayList de cantidades.

Para encontrar la posición, se creó una variable llamada índice, la cual almacenaría la posición del producto cuando este fuera encontrado. Durante la iteración con el for, se comparaba el nombre ingresado por el usuario con cada uno de los elementos del ArrayList de nombres. Para evitar errores relacionados con diferencias entre mayúsculas y minúsculas, se utilizó el método equalsIgnoreCase(), de esta forma: strNombres.get(i).equalsIgnoreCase(nombre). Este método permite hacer la comparación ignorando las diferencias de capitalización, lo cual mejora la experiencia del usuario.

Una vez encontrado el producto, era momento de actualizar la cantidad. Para ello, se utilizó el método set(), que permite modificar un valor en una posición específica dentro del ArrayList. Este método recibe dos parámetros: el índice donde se encuentra el valor a cambiar, y el nuevo valor que lo reemplazará. Por supuesto, este nuevo valor debía ser preguntado al usuario justo después de ingresar el nombre del producto, y antes de realizar la actualización.

Gracias a esta lógica y al uso del método set(), fue posible modificar correctamente la cantidad de cualquier producto que el usuario quisiera actualizar.

**Paso 4. cálculo del valor total para cada producto.**

Como es sabido, el programa debe calcular el valor total de cada producto, lo cual significa multiplicar la cantidad de cada producto por su precio unitario. Sin embargo, además de realizar esta operación, también es necesario almacenar estos valores totales en un array, ya que deben ser mostrados al final del programa.

Para esta parte del desarrollo, se creó un array común que almacenaría los valores totales. Este array tiene la misma dimensión que cualquiera de los otros tres ArrayLists ya existentes. Para obtener dicha dimensión se utilizó el método size(), el cual permite conocer el número de elementos de un ArrayList.

El cálculo de los valores totales se realizó mediante un ciclo for. Dentro de este ciclo, se tomó cada elemento del array de cantidades y se multiplicó por el elemento correspondiente del array de precios. El resultado de esta multiplicación se fue almacenando en el array de valores totales creado previamente.

Además, es importante mencionar que también se debe calcular el valor total del inventario. Este valor corresponde a la suma de todos los valores totales de los productos mencionados anteriormente. Para ello, dentro del mismo ciclo for en el que se calcularon los valores individuales, se fue sumando cada uno de ellos a una variable llamada "inventario", la cual se inicializó en cero y fue acumulando progresivamente el total de los productos.

**Paso 5. visualización final de datos.**

Se presenta la información de manera organizada usando un bucle for, mostrando nombre, cantidad, precio y valor total de cada producto. Finalmente, se imprime el valor total del inventario, sumando todos los productos.

CONCLUSIÓN

Este proyecto fue más que solo hacer que un programa funcionara. En realidad, fue todo un proceso de pensar, entender el problema y buscarle una solución paso a paso. Al principio no fue tan fácil, porque había que analizar bien qué era lo que se necesitaba y cómo íbamos a organizar todo para que el sistema hiciera lo que tenía que hacer. Fue ahí donde nos dimos cuenta de que, antes de empezar a programar, lo más importante era tener clara la lógica.

También aprendimos bastante al trabajar en grupo. A veces no todos pensábamos igual o no estábamos seguros de cómo seguir, pero con comunicación y escuchando ideas distintas pudimos avanzar. Dividirnos el trabajo, ayudarnos cuando algo no salía bien o cuando nos trabábamos, fue clave para poder sacar esto adelante.

Obviamente, durante el camino nos encontramos con varios errores y momentos en los que sentíamos que las cosas no salían como esperábamos. Pero justo en esos momentos fue donde más aprendimos. Corregir, intentar otra vez y no rendirse hizo la diferencia.

En resumen, más allá de lo técnico, este proyecto nos enseñó a pensar como personas que buscan soluciones, a trabajar con otros, a organizar mejor nuestras ideas y, sobre todo, a no tener miedo de equivocarnos en el proceso

# Referencias

1. Oracle. (n.d.). *Class ArrayList<E>*. Oracle Java Documentation. <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>
2. Oracle. (n.d.). *Class Scanner*. Oracle Java Documentation. <https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html>
3. Deitel, P. J., & Deitel, H. M. (2017). *Java: How to Program* (10th ed.). Pearson Education.
4. Eck, D. J. (2019). *Introduction to Programming Using Java* (Version 8.1). Hobart and William Smith Colleges. <http://math.hws.edu/javanotes/>
5. Liang, Y. D. (2021). *Introduction to Java Programming and Data Structures: Comprehensive Version* (12th ed.). Pearson.