Índice

1. Introducción **2**

2. Marco Teórico **2**

3. Desarrollo **3**

4. Resultados **5**

5. Conclusiones **6**

6. Reto para token **7**

1. Introducción

- **Planteamiento del Problema:** Se busca implementar un menú gráfico el cual permita añadir a una lista distintos objetos junto con su precio, así como eliminarlos de distinta manera, buscando simular el carro de compras de una tienda física o de aplicaciones en línea.
- **Motivación:** En esta práctica aprenderemos a encapsular objetos para protegerlos de modificaciones involuntarias o acceso a datos para brindarle mayor seguridad al programa, así como a empaquetar los códigos para una mayor organización y control.
- **Objetivos:** Implementar un programa que simule el carrito de compras de una tienda digital, mostrando en todo momento la lista completa de objetos dentro del carrito; encapsulando tanto el carrito como los artículos que contiene para controlar de mejor manera su obtención y adición, y empaquetando el código para una mayor organización y seguridad.

2. Marco Teórico

- **Encapsulamiento:** El *Encapsulamiento* es el principio de restringir el acceso directo a los atributos de una clase, de modo que solo puedan ser modificados mediante métodos controlados como getters y setters. Este enfoque nos ayuda a proteger los datos, evita inconsistencias y brinda mayor control del programa. [1]
- **Modificadores de acceso:** En java se utilizan distintos niveles de visibilidad para reforzar el encapsulamiento. Por ejemplo: el modificador *private* oculta los atributos de cualquier acceso externo a la clase, *public* los expone sin restricciones. Esta jerarquía de accesibilidad otorga flexibilidad al momento de diseñar clases más seguras y modulares. [1]
- **Métodos privados y abstracción:** El encapsulamiento también se extiende a métodos, que pueden declararse *private* si solo deben ser utilizados dentro de la propia clase. Esto nos permite ocultar cálculos internos o procedimientos auxiliares, exponiendo unicamente métodos públicos que definen la interfaz. [1]
- **Atributos (private final):** Decalarar un atributo como *private final* significa que es un dato privado al que solo es accesible dentro de la clase y es constante después de inicializare, debe recibir un valor en el momento de su declaración o en el constructor de la clase, y no puede modificarse posteriormente. [2]
- **Paquetes:** Los paquetes (también conocidos como "package") son una forma de organizar las clases relacionadas en espacios de nombres, de modo que el proyecto gane modularidad, claridad y una mejor estructura. Cada archivo fuente Java puede empezar con una línea *package nombredelpaquete;* indicando en qué paquetes se encuentra la clase. Esto permite acceder a esa clase desde fuera del paquete. [3]

3. Desarrollo

Librerías, clases, paso de mensajes y empaquetado de clases: Para esta práctica se implementó un sistema que simula un carrito de comprar con una interfaz gráfica. Para ello, se utilizaron dos librerías:

1. **javax.swing.*** : En este paquete se incluyen las clases necesarias para crear una interfaz gráfica de usuario, permite además crear ventanas, botones, listas y campos de texto.

2. **java.awt.event.*** : Proporcionan clases para manejar el diseño de la ventana y los eventos.

Todas las clases han sido organizadas o empaquetadas dentro de la carpeta: **/mx/unam/fi/poo/p56/**, al agregar: **package mx.unam.fi.poo.p56;** al inicio de cada clase.

La clase **Articulo** representa un objeto genérico del carrito, contiene los atributos **nombre** y **precio**. También se implementaron métodos getters para acceder a estos valores y un método **toString()** que devuelve una cadena con el nombre y el precio del artículo. De este modo, en la interfaz gráfica al agregar un artículo se mostrará en un formato especificado.

En la clase **Carrito** se utiliza un **ArrayList** para almacenar los objetos de la clase Articulo, cuenta con varios métodos:

- **agregar(Articulo a)** : Recibe un objeto de tipo Articulo, antes de agregarlo a la lista, verifica que no sea null y que el nombre no este vacío, de ser así devuelve false, en caso de ser válido se agrega a la lista con **articulos.add(a)** y devuelve true y se agrega a la lista.

- **eliminarPorIndice(int index)** : Recibe un número entero que representa la posición del artículo en la lista. Valida que el índice sea mayor o igual a 0 y que no se pase del tamaño actual de la lista con **articulos.size()**. Si el índice es válido, elimina el artículo en esa posición con **articulos.remove(index)** y devuelve true. Si el índice no es válido, devuelve false.

- **eliminarPorNombre(String nombre)** : Recibe una cadena que representa el nombre del artículo a eliminar. Valida que el nombre no sea null ni vacío. Recorre la lista completa con un ciclo **for** y compara el nombre de cada artículo con el nombre ingresado, ignorando mayúsculas y minúsculas con **equalsIgnoreCase**. Si encuentra coincidencia, utiliza **remove(i)** para eliminar ese artículo de la lista y devuelve true, de lo contrario devuelve false.

- **limpiar()** : Elimina todos los artículos de carrito usando **articulos.clear()**.

- **getArticulos()** : Este devuelve una copia de la lista de artículos, protegiendo así a la lista original de modificaciones externas.

- **getTotal()** : Recorre todos los artículos en la lista sumando sus precios al utilizar un ciclo **for** para acumular el valor de cada artículo, para ello, hace uso del método **getPrecio** de la clase Articulo, por último devuelve el precio total.

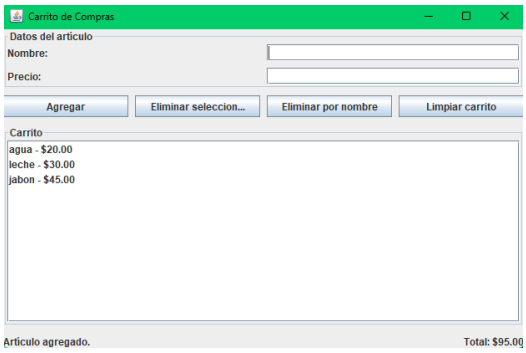
Finalmente la clase **MainApp** que contiene el metodo main, comienza creando un objeto de la clase Carrito para almacenar los artículos y otro de la clase Vista para representar la ventana de una aplicación. Posteriormente mediante listeners, cada botón de la vista se conecta con la lógica del carrito:

- **Botón agregar** : Toma los valores escritos en los campos de texto, crea un nuevo Articulo, lo agrega al carrito y lo muestra en la lista de la ventana.
- **Botón Eliminar seleccionado** : Detecta el artículo marcado en la lista y lo elimina tanto de la vista como del carrito.
- **Botón Eliminar por nombre** : Pide al usuario el nombre de un artículo mediante un cuadro de diálogo y elimina los artículos que coincidan.
- **Botón Limpiar carrito** : Vacía completamente el carrito y actualiza la interfaz de la vista.

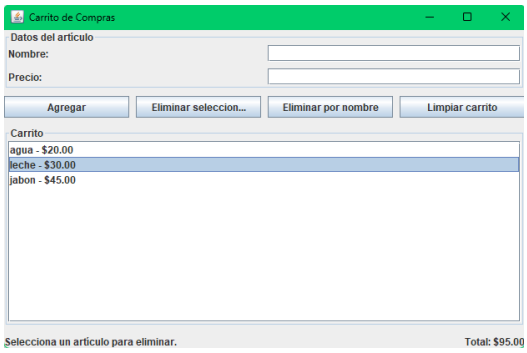
4. Resultados

```
javac -d . .\mx\unam\fi\poo\p56\*.java
java mx.unam.fi.poo.p56.MainApp
```

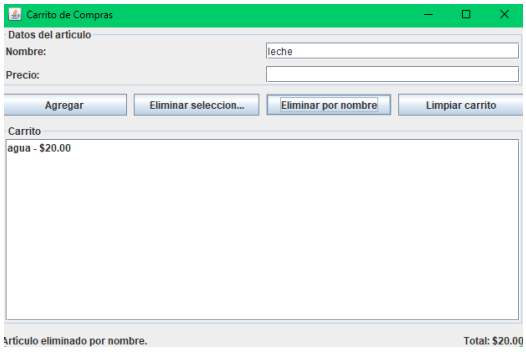
Compilación y ejecución usando paquetes



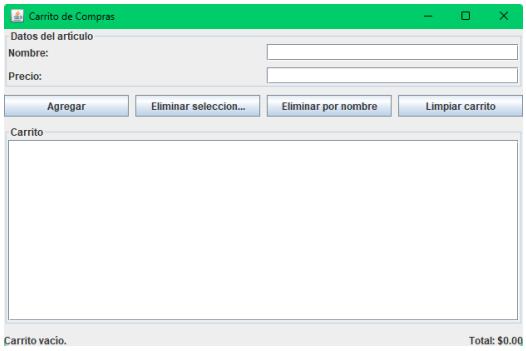
Agregar artículos al carrito



Eliminar artículos por selección



Eliminar artículos por nombre



Vaciar carrito

5. Conclusiones

A través de esta práctica, logramos consolidar nuestros conocimientos sobre el manejo de objetos en Java. Pusimos especial atención a la encapsulación, asegurándonos de que los atributos de las clases estuvieran protegidos y solo se accediera a ellos mediante métodos designados. Además, también comprendimos sobre la relevancia del empaquetamiento como estrategia clave para una buena organización del código y su posterior mantenimiento. La simulación del carrito de compras sirvió como un excelente ejemplo de pruebas, pues nos permitió implementar operaciones básicas como la adición, eliminación y visualización de artículos en un entorno gráfico. Definitivamente, esta experiencia consolidó nuestra visión sobre el diseño modular en Java y la importancia de una interacción segura entre clases para crear aplicaciones más estructuradas y funcionales.

Referencias

[1] *Introducción a POO en Java: Encapsulamiento*. 2023. URL: `https://openwebinars.net/blog/introduccion-a-poo-en-java-encapsulamiento/`.

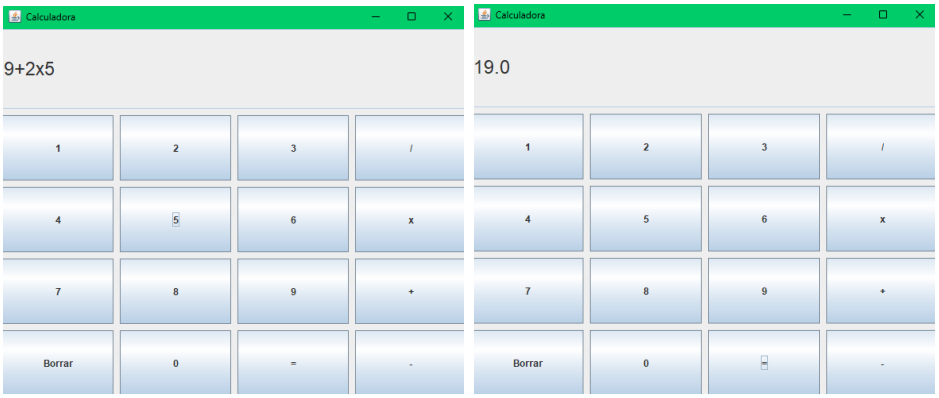
[2] *Private and Final Methods in Java*. 2025-04-24. URL: `https://www.geeksforgeeks.org/java/private-and-final-methods-in-java/`.

[3] *Introducción a POO en Java: Interfaces y paquetes*. 2023-09-22. URL: `https://openwebinars.net/blog/introduccion-a-poo-en-java-interfaces-y-paquetes/`.

6. Reto para token

Para realizar este reto, reutilizamos las clases **Vista** y **MainApp**, cambiando los botones y los frames, que es donde se muestra el texto. En la clase **MainApp** cambiamos los listeners, poniendo que pasa en cada caso, utilizamos listas que guardaran tanto los números como las operaciones, y al darle igual, se calculara lo que hay dentro de esas listas.

Utilizamos banderas para resolver los errores que nos salían en el programa.



Ejecución de la calculadora

	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación
Salas A y B

Profesor(a):

Asignatura:

Grupo:

No de Práctica(s): Practica 5-6

Integrante(s): 322246320

322267567

322012051

322236688

322330872

No. de lista o brigada:

Semestre

Fecha de entrega:

Observaciones:

CALIFICACIÓN: _____