	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): Dávila Pérez René Adrián

Asignatura: Programación Orientada a Objetos

Grupo: 1

No de Práctica(s): 5 y 6

Integrante(s): 322089020

322089817

322151194

425091586

322085390

No. de lista o brigada: Equipo 4

Semestre: 2026-1

Fecha de entrega: 3 de octubre de 2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

1.1. Planteamiento del Problema

El ejercicio consiste en refactorizar una aplicación Java existente para alinearla con los principios fundamentales de la Programación Orientada a Objetos. El reto principal es aplicar correctamente el **encapsulamiento** de datos, de manera específica en la clase `Artículo`, para proteger su estado interno. Adicionalmente, se requiere organizar la totalidad del código fuente dentro de una estructura de **paquetes** definida por el *Full Qualified Name* `mx.unam.fi.poo.p56`, asegurando que la aplicación mantenga su funcionalidad original tras las modificaciones.

1.2. Motivación

Esta práctica busca consolidar los conceptos teóricos de encapsulamiento y paquetes vistos en clase, llevándolos a una implementación concreta. La motivación es comprender en un escenario práctico cómo estas técnicas no solo organizan el código, sino que también mejoran su mantenibilidad, robustez y seguridad. Al aplicar estos principios, se demuestra su valor en el desarrollo de software escalable y profesional.

1.3. Objetivos

El objetivo principal de esta práctica es aplicar adecuadamente los conceptos de encapsulamiento y paquetes para mejorar la estructura y calidad de una aplicación Java.

- Implementar el encapsulamiento en la clase `Artículo`, declarando sus atributos como privados y proveyendo métodos de acceso públicos (getters y setters).
- Estructurar el código fuente de la aplicación bajo el paquete `mx.unam.fi.poo.p56`.
- Asegurar que la interacción entre el modelo de datos (clases `Artículo` y `Carrito`), la `Vista` y el controlador (`MainApp`) funcione correctamente después de la refactorización.

- Documentar la solución teórica y la implementación práctica para reforzar el aprendizaje obtenido en el curso.

2. Marco Teórico

La Programación Orientada a Objetos (POO) es un paradigma de desarrollo de software que organiza el código en torno a entidades llamadas *objetos*, los cuales representan elementos del mundo real o abstracto. Estos objetos combinan datos (atributos) y comportamientos (métodos), favoreciendo la modularidad, la reutilización y la escalabilidad de los sistemas [1]. Entre los pilares fundamentales de la POO se encuentran la **abstracción**, la **herencia**, el **polimorfismo** y la **encapsulación**, siendo este último el de mayor relevancia en el presente trabajo.

2.1. Encapsulación

La encapsulación consiste en restringir el acceso directo a los atributos internos de un objeto, permitiendo que cualquier modificación o consulta se realice mediante métodos definidos específicamente para ello [2]. Esta técnica proporciona un mecanismo de protección de datos, de modo que el estado de un objeto no pueda alterarse de manera indebida, garantizando así mayor seguridad e integridad en el programa.

En Java, la encapsulación se logra declarando los atributos como `private` y ofreciendo acceso controlado a través de métodos `getter` y `setter`. Esto permite aplicar validaciones antes de modificar un atributo o limitar la visibilidad de la información sensible.

2.2. Paquetes

Un **paquete** en Java es un contenedor lógico que agrupa clases, interfaces y subpaquetes relacionados entre sí, funcionando como un mecanismo de organización y encapsulación a nivel superior. Su uso facilita la estructuración del código en proyectos grandes, al tiempo

que evita conflictos de nombres y mejora la claridad del diseño [1].

Los paquetes se definen con la palabra reservada `package`, seguida de un identificador jerárquico separado por puntos. Por ejemplo:

```
package mx.unam.fi.poo.p56;
```

Este esquema jerárquico refleja la procedencia y propósito del código, además de constituir una convención estándar en entornos académicos y profesionales. En el caso de Java, la propia biblioteca estándar está organizada en múltiples paquetes, como `java.lang`, `java.util` o `java.io`, que contienen las herramientas fundamentales para el desarrollo.

2.3. Relación con la práctica

En el presente proyecto, la encapsulación se aplica para proteger los atributos de la clase `Artículo`, de modo que solo puedan ser accedidos mediante métodos de acceso controlado. Al mismo tiempo, la organización en el paquete `mx.unam.fi.poo.p56` busca reflejar una estructura ordenada y coherente, similar a la empleada en aplicaciones profesionales. De esta manera, se demuestra cómo los principios teóricos de la POO pueden trasladarse directamente a la implementación práctica, reforzando la importancia de una correcta organización y resguardo de datos en el desarrollo de software.

3. Desarrollo

3.1. Implementación del Código

3.1.1. Articulo

Define qué es un ***articulo***, este representa un único artículo que se puede comprar. Sus atributos son ***nombre*** del artículo ***precio*** del artículo. Esta clase usa métodos ***getters*** y ***setters*** que son métodos públicos que permiten acceder y modificar de forma controlada los atributos ***nombre*** y ***precio***, que están encapsulados con ***private***. Por último el método ***toItemString()***, este convierte la información del ***artículo*** (***nombre*** y ***precio***) en un ***String*** legible para el usuario.

3.1.2. Carrito

Gestiona la colección de artículos de compra. Utiliza una lista privada para almacenar los objetos ***Articulo***, encapsulando los datos y controlando el acceso a través de métodos públicos. Esta clase maneja todas las operaciones fundamentales del carrito: permite ***agregar*** artículos (con validaciones para evitar datos nulos o vacíos), ***eliminar*** artículos específicos ya sea por su posición en la lista o por su nombre, y ***vaciar*** completamente el carrito con el método ***limpiar***. Además, es responsable de calcular el ***costo total*** de la compra sumando los precios de todos los artículos y proporciona una copia segura de la lista de artículos para ser mostrada en la interfaz.

3.1.3. MainApp

Contiene el punto de entrada y la lógica de control de la aplicación. Su función es instanciar los objetos del ***Carrito*** y de la ***Vista***, y posteriormente acoplar la interacción del usuario con la lógica de negocio. Esto se logra mediante la asignación de ***ActionListeners*** a los componentes Swing de la ***Vista***. Cada ***ActionListener*** es un manejador de eventos que se ejecuta en el Event Dispatch Thread (EDT) a través de ***SwingUtilities.invokeLater***. Al

activarse un evento, el código del listener extrae los datos de los componentes de entrada de la ***Vista***, realiza validaciones, invoca los métodos correspondientes en la instancia de ***Carrito*** para manipular el estado de la aplicación y, finalmente, actualiza la ***Vista*** para que su representación visual sea consistente con el estado actual del modelo.

3.1.4. Vista

Define y encapsula la totalidad de la interfaz gráfica de usuario (GUI). La clase hereda de ***javax.swing.JFrame***, sirviendo como el contenedor de la ventana principal de la aplicación. Dentro de su constructor, se instancian y configuran todos los componentes Swing requeridos, incluyendo ***JTextField*** para la entrada de datos, ***JButton*** para las acciones del usuario, y una ***JList*** asociada a un ***DefaultListModel*** para mostrar la lista de artículos. La disposición visual de estos componentes se gestiona mediante el uso de contenedores ***JPanel*** y gestores de diseño como ***BorderLayout*** y ***GridLayout***. Todos los componentes de la UI son declarados como campos privados, y el acceso a ellos desde clases externas (como el controlador) se concede únicamente a través de métodos ***getter*** públicos. Este diseño permite al controlador registrar ***ActionListeners*** y manipular el estado de la vista sin violar su encapsulamiento.

3.2. Pruebas

3.2.1. Agregar producto

```
Output(En ventana): Datos del artículo
Nombre:
Precio:
Input(Args):
Pan integral
45.5
Output(En ventana): Carrito
Pan integral - $45.50
```

3.2.2. Eliminar selección

Output(En ventana):

Agregar | Eliminar selección | Eliminar por nombre | Limpiar carrito

Carrito

Pan integral - $45.50

Leche deslactosada 1 L -$35.00

Galletas de arroz -$40.00

Input(Args):

Seleccionamos "Galletas de arroz -$40.00"

Seleccionamos 'Eliminar por selección'

Output(En ventana)

-- Artículo eliminado por selección

3.2.3. Eliminar por nombre

Output(En ventana):

Agregar | Eliminar selección | Eliminar por nombre | Limpiar carrito

Carrito

Pan integral - $45.50

Leche deslactosada 1 L -$35.00

Galletas de arroz -$40.00

Input(Args):

Escribimos "Galletas de arroz - $40.00" en la casilla nombre

Seleccionamos 'Eliminar por nombre'

Output(En ventana):

-- Artículo eliminado por nombre

3.2.4. Limpiar carrito

Output(En ventana):

Agregar | Eliminar selección | Eliminar por nombre | Limpiar carrito

Carrito

Pan integral - $45.50

Leche deslactosada 1 L -$35.00

Galletas de arroz -$40.00

Input(Args):

Seleccioinamos 'Limpiar carrito'

Output(En ventana):

-- Carrito vacío

4. Resultados

```
carlo@Santantango:~/POO/Practicas/Practica56$ javac *.java
carlo@Santantango:~/POO/Practicas/Practica56$ java MainApp
```

Figura 1: Se compila y ejecuta el código principal

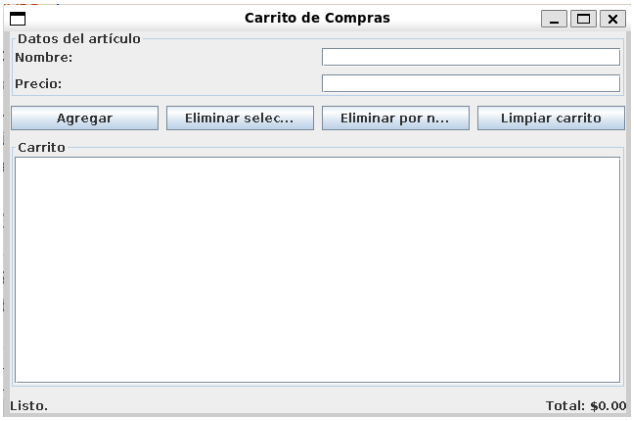

Figura 2: Saldrá una ventana con todas las posibles acciones

Figura 3: Se ingresa un elemento al carrito

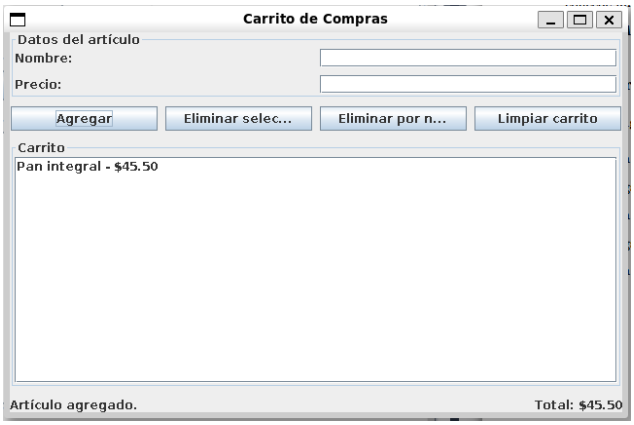

Figura 4: El producto se muestra agregado al carrito

Figura 5: Se selecciona un producto de la lista y se elige la opción Eliminar selección

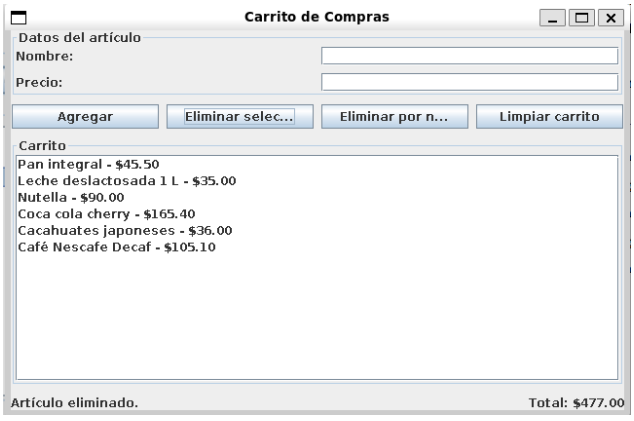

Figura 6: Se muestra como el elemento anteriormente seleccionado ya no se encuentra en la lista

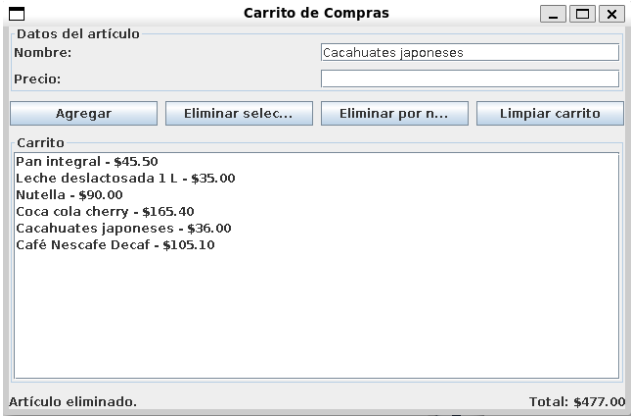

Figura 7: En la casilla de nombre se escribe el nombre correspondiente al producto que se desea eliminar

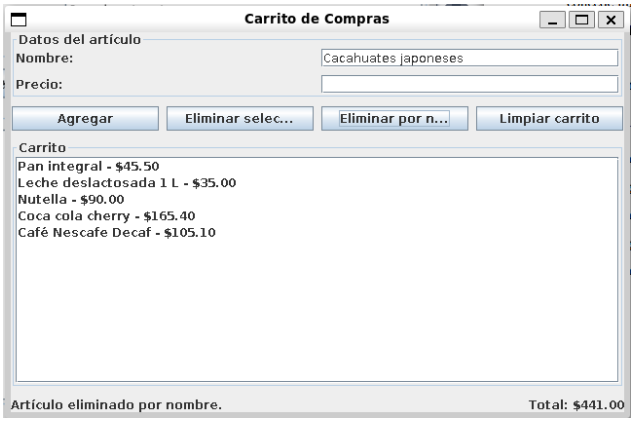

Figura 8: El producto anteriormente nombrado ya se eliminó de la lista

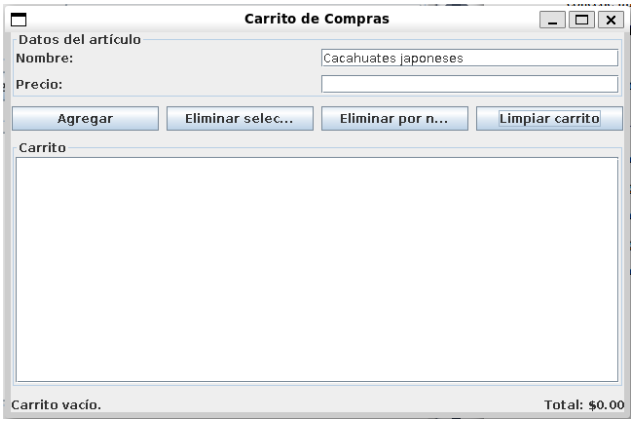

Figura 9: La opción Limpiar carrito elimina todos los productos que estaban en el carrito

5. Conclusiones

El desarrollo de esta práctica permitió aplicar de manera efectiva los conceptos de **encapsulamiento y organización por paquetes**, cumpliendo con todos los objetivos planteados. Se logró refactorizar la aplicación de manera que la integridad de los datos de la clase `Artículo` quedara protegida, demostrando la importancia de ocultar la implementación y exponer únicamente una interfaz controlada para su manipulación.

La reestructuración del proyecto en el paquete `mx.unam.fi.poo.p56` evidenció los beneficios de tener un espacio de nombres bien definido, lo cual previene conflictos y mejora la navegabilidad y el mantenimiento del código a largo plazo. Esta práctica reafirma que la correcta aplicación de los pilares de la Programación Orientada a Objetos no es un ejercicio meramente teórico, sino un requisito indispensable para la construcción de software funcional, robusto y profesional.

Referencias

[1] Jorge Martínez Ladrón de Guevara. "Fundamentos de programación en Java". En: *Editorial EME* (2011).

[2] Oracle. *Encapsulation in Java*. Accedido el 2 de octubre de 2025. 2023. URL: `https://docs.oracle.com/javase/tutorial/java/concepts/`.