	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): René Adrián Dávila Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 7

No de Práctica(s): 5 y 6

Integrante(s): 322044339

322094152

322114461

425093384

322150984

No. de brigada: 6

Semestre: 2026-1

Fecha de entrega: 01 de octubre de 2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

Comúnmente, la exposición directa de los atributos de una clase implica la modificación errónea de información, estados inconsistentes o complicaciones al dar mantenimiento al código. Esto se vuelve super importante en aplicaciones que usan datos sensibles o información que cambia frecuentemente, como un carrito de compras, en donde los precios o los artículos pueden ser modificados sin autorización; cosa que puede generar errores de los que no sepamos su origen y, con ello, que el usuario pierda nuestra confianza.

Es por ello que la clave para este problema es el encapsulamiento [1], que surge de la necesidad de esconder los detalles internos de una clase y que, además, controla el acceso a través de métodos específicos. Pensemos que tenemos una caja fuerte: los objetos valiosos en su interior son los atributos, mientras que los getters y setters son la combinación que abre la caja y que solo quienes conocen el código pueden usar. De esta manera, se protege la integridad de los datos y se facilita la escalabilidad y el trabajo colaborativo.

El objetivo de esta práctica es aplicar el encapsulamiento en la clase Articulo y organizar el código en el paquete mx.unam.fi.poo.p56, manteniendo la funcionalidad original de la aplicación. Con esto, se busca reforzar el uso de este principio de la programación orientada a objetos y comenzar a implementarlo para desarrollar un código más robusto, modular y fácil de mantener.

2. Marco Teórico

2.1. Encapsulamiento

Es el proceso de empaquetamiendo de las variables de un objeto con la protección de sus métodos, es un concepto utilizado en POO por dos beneficios principales que aporta está práctica:

- Modularidad

  Permite escribir o dar mantenimiento a un objeto de forma que no afecte otras partes del proyecto que utilizan dicho objeto.

- Ocultar información

  Esto significa, que podemos controlar el acceso a diferentes partes del código, ocultando la información que queremos que no sea fácil de manipular en nuestro código.[3]

Al momento de poner en práctica este concepto, es necesario encontrar alguna manera de interactuar con las propiedades de los objetos que se encuentran encapsuladas, fuera de la clase que los contiene.

- Métodos setter

  Establece o actualiza el estado de las propiedades encapsuladas.

- Métodos getter

  Retorna el valor contenido actualmente en las propiedades encapsuladas.

2.2. Empaquetado

El empaquetado en Java es el proceso de organizar y distribuir aplicaciones en archivos estandarizados que facilitan su despliegue. Una aplicación Java EE se entrega típicamente en un archivo JAR (Java Archive), WAR (Web Archive) o EAR (Enterprise Archive). Los archivos WAR y EAR son extensiones de JAR estándar, con sufijos .war o .ear, lo que permite reutilizar componentes comunes sin código adicional, solo ensamblando módulos en estos formatos.

Un archivo EAR agrupa varios módulos Java EE y, opcionalmente, descriptores de despliegue. Estos descriptores son documentos XML que definen configuraciones como atributos de transacción, autorizaciones de seguridad, sin necesidad de alterar el código fuente. La información de despliegue se especifica principalmente mediante anotaciones en el código, pero los descriptores (archivos que contiene metadatos que indican el comportamiento que debería tener la aplicación) las sobrescriben si están presentes. Existen dos tipos: los descriptores Java EE, estandarizados por la especificación para cualquier implementación compatible, y los de runtime, específicos del servidor.[2]

3. Desarrollo

En esta práctica realizamos varias cosas sobre los códigos proporcionados, esto viene siendo desde volver a utilizar la Graphical User Interface para un entorno más realista y concreto, la utilización y manejo de paquetería lo cual abre las puertas a realizar ecosistemas de documentos de una forma más avanzada, y también el uso de encapsulamiento que implica el importante concepto de seguridad, lo cual es uno de los pilares indispensables cuando estamos programando. Para ello, desglosaremos cada clase y método, priorizando el concepto de encapsular.

3.0.1. Paquetería

Antes de siquiera empezar a encapsular la práctica optamos por ir empaquetando nuestros documentos.java en una dirección mx/unam/fi/poo/p56. A todos nuestros archivos le pusimos que importaran package mx.unam.fi.poo.p56, y ahí mismo los almacenamos.

Posteriormente a almacenarlos tuvimos el objetivo de compilarlo, pero al ser paquete cambia el como debemos de compilar, pudiendo hacerlo con javac mx/unam/fi /poo/p56/*.java, si estamos en la raíz del proyecto o para compilarlo recursivamente con -d .mx/unam/fi/poo/p56/*.java. Al momento de compilar usando el primer

método nos dio un error aunque al parecer este puede ser pasado por alto por el momento.

3.0.2. Articulo.java

Ahora adentrandonos más en el código, la clase Articulo se dedica solamente a crear objetos artículos almacenando el nombre del producto y su precio. Después del método constructor tenemos un método que retorna una cadena con el nombre y el precio del artículo pero separando la moneda en pesos y centavos, redondeandolos al multilpicarlos por cien, diviendolos entre 100 y finalmente obteniendo el módulo.

Si es que el usuario añade un producto como agua y quiere que tenga un precio de 20.500 pesos, entonces precisamente el programa arrojará Agua - $ 20.50

En esta clase no hay getters y setters visibles.

3.0.3. Carrito.java

La función principal de la clase carrito es manejar precisamente cuantas y cuales productos tenemos registrados. Está clase tiene operaciones importantes, siendo estas, agregar, que agrega un objeto al carrito si su nombre no es null, eliminar por indice, que elimina artículos con solo mandar su posición, eliminar por nombre, elimina al primer articulo que tenga el mismo nombre, limpiar, vacía todo el carrito, getArticulos, que implica ya el uso de get devuelve una copia de la lista de artículos, y getTotal.

Si bien no es precisamente privado, controlamos su acceso desde otras clases mediante un getter en getArticulos, por lo que podemos decir que está protegida y solo carrito puede modificarse así mismo mediante el resto de sus operaciones

3.0.4. Vista.java

Esta clase está especializada en la creación de la interfaz gráfica GUI, esta es mucho más extensa y compleja que las vistas anteriormente, consta de dos campos de entrada, uno para el nombre, otro para el precio, cuatro botones para agregar, eliminar por indice, eliminar por nombre y para limpiar el carrito, etiquetas y una lista para mostrar los artículos. Utilizamos un get para getNombreField, esto hace que la clase principal los use para procesar o asignar eventos, haciendo de la clase Vista encargada solo de la parte gráfica. Al final utilizamos una gran cantidad de gets para utilizar métodos con funciones como el obtener nombres, precios, agregar botones, eliminar por indice o por nombre, limpiar el carrito, obtener la lista y el uso de etiquetas.

3.0.5. Main.java

Esta es la clase principal dado que contiene al método main. El comando SwingUtiles.invokeLater(() ->{}, se dedica al apartado gráfico y manejo de botones, este no tiene excepciones por ende no requiere de try/catch. Después utilizamos los listeners, que se encargan de escuchar eventos y procesarlos para un correcto funcionamiento

del programa. Prevemos situaciones que pudieran dañar o crear confusión en le programa como no admitir nombres nulos o formatos incorrectos en precios de productos. Creamos objetos Articulo con los que podemos operar los métodos mencionados con anterioridad, esto usando get para obtener estados listas nombres, etc. Esta clase solo posee métodos estáticos por lo que directamente no aplica encapsulamineto, sin embargo, al ser la principal hace mucho uso de get para obtener la información de los atributos privados que si tienen otras clases, por lo que hay un transito controlado y seguro de datos.

4. Resultados

4.1. Carrito de la compra

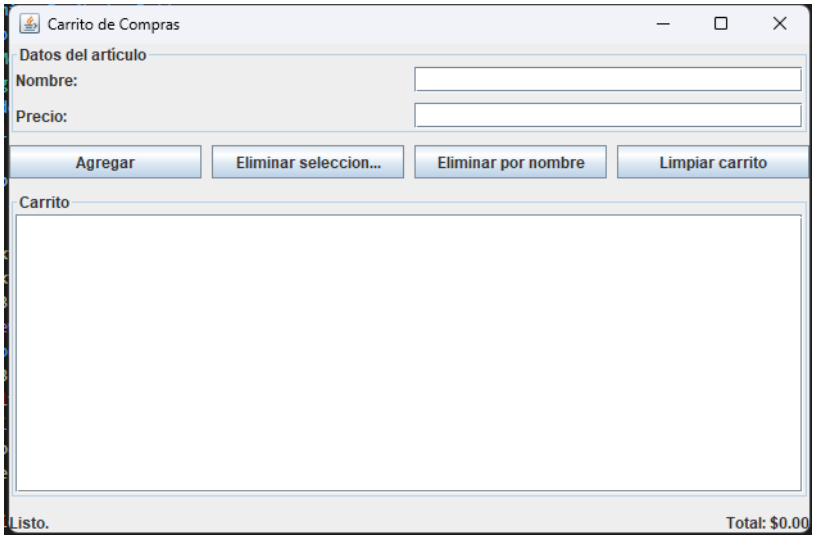

Figura 1: Interfaz inicial

Una vez que compilamos y ejecutamos el MainApp, se muestra la seguiente ventana emergente que nos permite interactuar con nuestro carrito de la compra.

4.2. Ingresar artículos

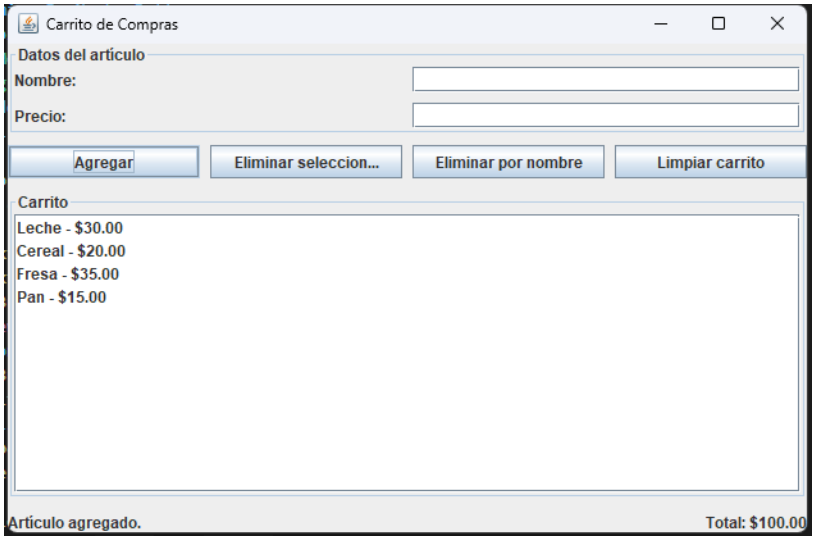

Figura 2: Ingreso de 4 artículos

Ingresamos el nombre y el precio (con números) de cuatro artículos y los añadimos al carrito, en la parte inferior derecha se nos muestra el precio acumulado.

4.3. Eliminar por selección

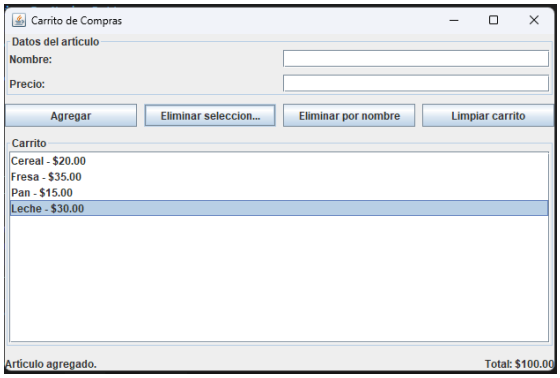

Figura 3: Artículo "Leche" seleccionado

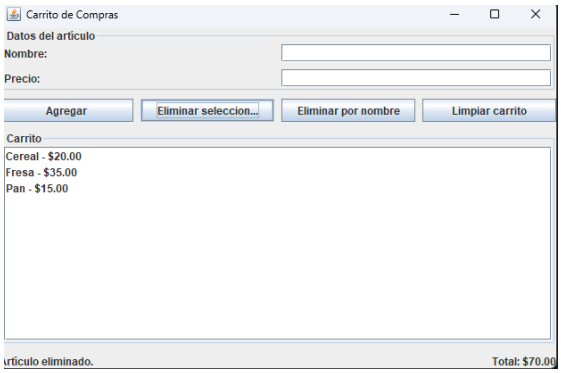

Figura 4: Artículo "Leche" eliminado

Cuando se selecciona uno de los artículos ya ingresados y posteriormente se elige la opción "Eliminar por selección", el artículo se elimina del carrito y el precio acumulado se actualiza.

4.4. Eliminar por nombre

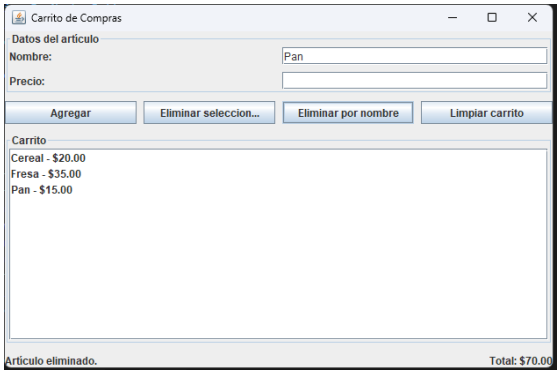

Figura 5: Nombre "Pan" ingresado

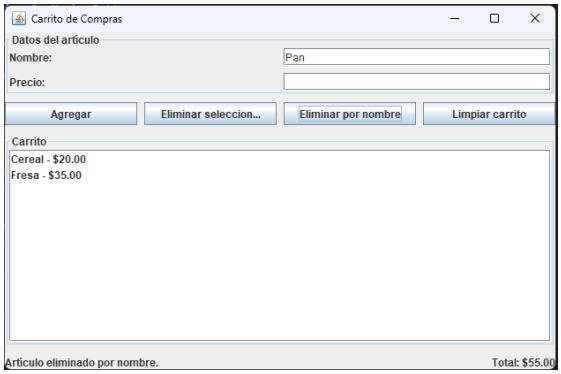

Figura 6: Artículo "Pan" eliminado

Cuando ingresamos el nombre de un artículo y elegimos la opción de "Eliminar por nombre", busca si hay un artículo con ese nombre, y en caso de haberlo, lo elimina y actualiza el precio acumulado, en caso contrario, mostrará un mensaje de error.

4.5. Limpiar carrito

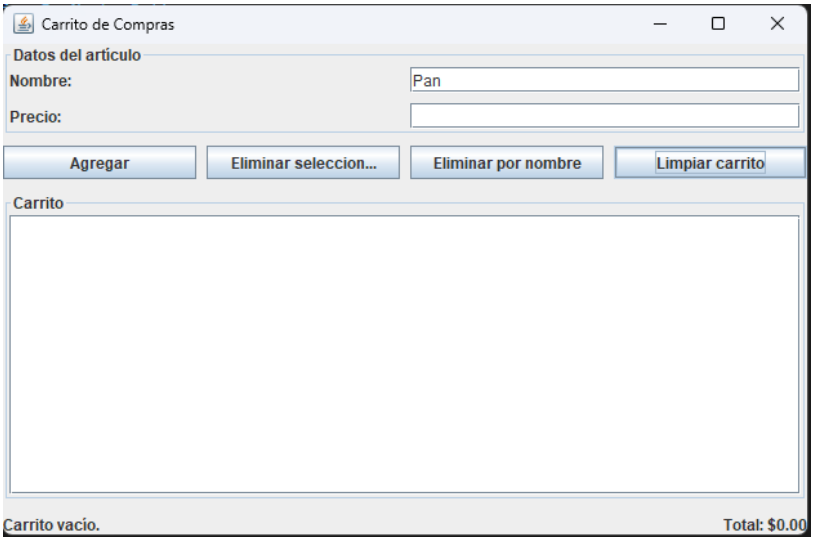

Figura 7: Carrito vacío

Cuando seleccionamos la opción de "Limpiar carrito", todos los artículos ingresados se eliminan y el precio acumulado se restablece en cero.

4.6. Precio inválido

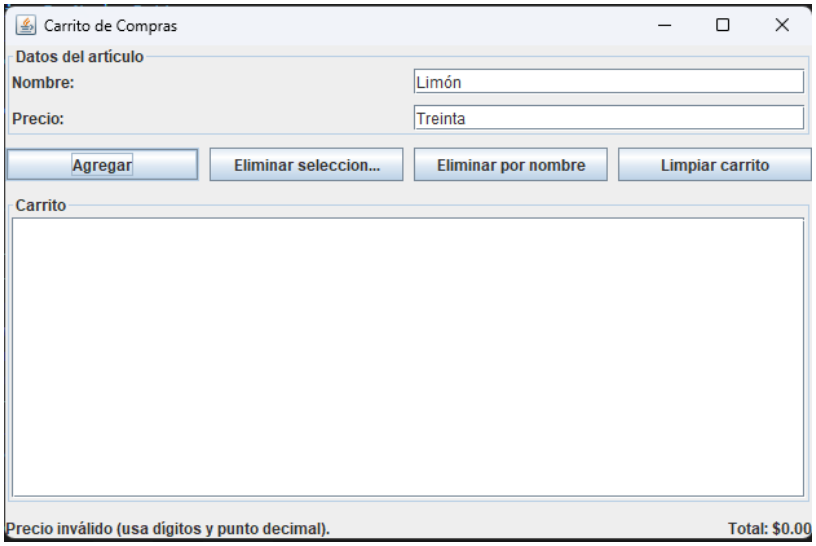

Figura 8: Mensaje de precio inválido

Cuando intentamos ingresar un artículo nuevo, pero ingresamos el precio en un formato diferente al "Número.Decimal" el programa nos muestra un mensaje de precio inválido.

4.7. Finalización del programa

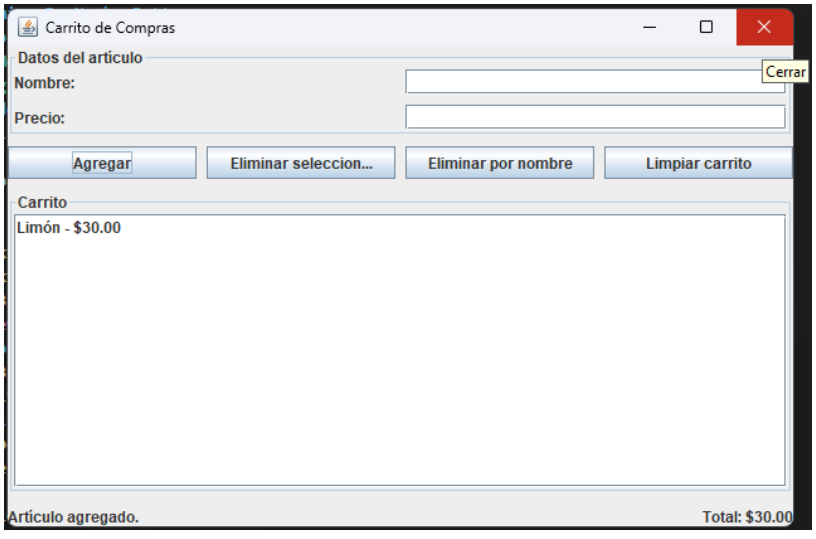

Figura 9: Seleccionando la opción "cerrar"

Una vez que seleccionamos la opción para cerrar la ventana emergente, el programa finaliza.

5. Conclusiones

La práctica permitió comprobar cómo la correcta aplicación del encapsulamiento y del empaquetado facilita la organización del código y el control de acceso a los datos. Al observar el funcionamiento del carrito de compras con y sin encapsulamiento, nos dimos cuenta de su idéntico funcionamiento a pesar de que restringimos el acceso a sus atributos. Aunque es difícil entender el encapsulamiento a tan pequeña escala con códigos simples, se entiende la necesidad de restringir el acceso directo a atributos en programas a gran escala por motivos de seguridad, evitar inconsistencias y permitir operaciones seguras.

De nuevo, en esta práctica reforzamos el concepto de separación de responsabilidades, esta vez con la aplicación de paquetes y el mantenimiento de nuestro código de forma modular.

6. Referencias bibliográficas

Referencias

[1] Jorge López Blasco. *Introducción a POO en Java: Encapsulamiento*. Accedido: 1 de octubre de 2025. 2023. URL: https://openwebinars.net/blog/introduccion-a-poo-en-java-encapsulamiento.

[2] Oracle Corporation. *Packaging Web Components*. 2013. URL: https://docs.oracle.com/javaee/7/tutorial/packaging001.htm (visitado 10-2023).

[3] Facultad de Ingeniería, UNAM. *Encapsulamiento*. 2023. URL: http://profesores.fi-b.unam.mx/carlos/java/java_basico3_3.html (visitado 10-2023).