Índice

1. Introducción

El problema a resolver en esta práctica es el desarrollo de una aplicación de escritorio que simula un carrito de compras, utilizando el lenguaje Java. La motivación principal es aplicar y comprender la importancia de conceptos clave de la Programación Orientada a Objetos, como lo son el encapsulamiento y los paquetes. El encapsulamiento es muy importante para proteger la información y la implementación de los objetos, mientras que los paquetes son esenciales para organizar las clases y evitar conflictos de nombres. Realizar esta práctica permite complementar y mejorar lo trabajado en el proyecto anterior, además de reforzar el uso de los conceptos vistos, a través de este problema que representa un caso práctico.

1.1. Objetivos

- Encapsular el programa, mediante atributos privados y los métodos *getter* y *setter*.
- Estructurar el código fuente de la aplicación dentro de un paquete con el Full Qualified Name *mx.unam.fi.poo.p56*.

2. Marco Teórico

- **Encapsulamiento**:
  Característica de la Programación Orientada a Objetos que refiere a la práctica de agrupar en una sola unidad (una clase) tanto los datos (atributos) que describen el estado de un objeto como los métodos que operan sobre esos datos y sólo pueden accederse a ellos a través de métodos públicos de la clase. [8]
  El principio clave del encapsulamiento es el ocultamiento de la información, que se logra declarando los atributos como privados (private). Esto impide que sean accedidos o modificados directamente desde fuera de la clase. [5]
  - **Getter**: Método que permite la recuperación de un atributo de una clase en Java.[8]
  - **Setter**: Método que permite la asignación del valor de un atributo de una clase en Java, debe realizar la verificación de que la asignacion es correcta dentro del dominio del problema.[8]
- **Paquetes (Packages)**:
  En Java, un paquete es un mecanismo para organizar clases e interfaces relacionadas en un espacio de nombres (namespace) común. Su función es análoga a la de las carpetas en un sistema de archivos. Los paquetes son cruciales para: [6]
  - Organización: Agrupan clases con funcionalidades similares (por ejemplo, java.util contiene clases de utilidades como ArrayList).

- Prevención de Conflictos de Nombres: Dos clases pueden tener el mismo nombre siempre que pertenezcan a paquetes diferentes. El nombre completo de una clase, o Full Qualified Name (FQN), incluye su paquete.
- Control de Acceso: Los paquetes se usan en conjunto con modificadores de acceso (public, protected, private y default) para controlar qué clases pueden acceder a otras.

- **Final**:
  En Java, la palabra reservada final es un modificador que se utiliza para declarar que una entidad (una variable, un método o una clase) no puede ser modificada o alterada después de su definición inicial. [3]

- **Clases**: son modelos o abstracciones de la realidad que representan un elemento de un conjunto de *objetos* [1]

- **Objeto:** es la estructura o entidad abstracta que se crea a partir de la definición de una *clase*, de tal manera, que ésta permite organizar y relacionar los *datos* de un programa o problema (*atributos*) con sus respectivos *métodos*. [2]

  - **Atributos**: Conjunto de rasgos o características que posee un objeto [2]
  - **Métodos**: Conjunto de procedimientos o funciones para la manipulación o interacción con los datos de un objeto.[2]
    - **Constructor:** Método que tiene el mismo nombre que la clase y cuyo propósito es inicializar los atributos de un nuevo objeto. Se ejecuta automáticamente cuando se crea un objeto o instancia de la clase.[9]
    - **Método estático:** métodos que no requieren un objeto para ser invocados, se pueden ejecutar a través de la clase. [9]
    - **Método de instancia:** Estos métodos operan en instancias de una clase y pueden acceder a los campos y otros métodos de esa instancia. Son invocados a través de un objeto creado a partir de la clase. [4]

- **ArrayList** [7]
  Es una clase que implementa la interfaz List y proporciona métodos para manipular el tamaño de la matriz que se utiliza internamente, permitiendo almacenar elementos de forma dinámica.

  Sintaxis:
  ArrayList<String>nombreArrayList= new ArrayList<String>();
  (puede ser de cualquier otro elemento u objeto).

  Métodos implementados

  - **add**: Anexa el elemento especificado al final de la lista.
  - **remove**: Quita el elemento en la posición especificada en la lista.
  - **get**: Devuelve el elemento en la posición especificada en la lista.

3. Desarrollo

1. **Clase Articulo:**
   Para la clase Artículo, se les asignó el modificador de acceso **private** a las variables de nombre y precio. Para poder interactuar con dichas variables creamos los getters y setters para cada una (con modificador de acceso public), de forma que sea posible realizar operaciones de lectura y escritura de datos. Los getters y setters implementados fueron:

   - getNombre
   - setNombre
   - getPrecio
   - setPrecio: este método contiene una validación para aceptar únicamente valores positivos.

   Finalmente, agregamos un constructor, el cual inicializa el nuevo objeto de la clase Artículo con los valores específicos que cumplan con las características de esta clase.

2. **Clase Carrito:**
   En la clase Carrito, el encapsulamiento tambien tuvo que ser implementado. Para la lista de articulos, se declaro con los modificadores de acceso **private final**, para que solo pudiera ser manipulada por los métodos internos de la clase. Para el constructor unicamente tuvimos que añadir la llamada a los getters correspondientes para el que los métodos de agregar y eliminarPorNombre funcionaran correctamente debido a los cambios realizados en el acceso a estos datos.

3. **Clase MainApp:**
   Esta clsase conecta la clase Vista con la clase Carrito, llamando a sus métodos de la siguiente manera en el método main:

   - Se instancian los objetos Carrito y Vista.
   - Se configuran los ActionListener para cada uno de los botones de la interfaz. Cada listener define el comportamiento que se debe ejecutar al hacer clic:
     - Agregar: lee el nombre y precio, crea un nuevo objeto Articulo y lo pasa al método carrito.agregar().
     - Eliminar: Lee la selección del usuario o el texto del campo del nombre y llama al método que corresponda en *Carrito*.
     - Después de cada función que modifica el estado del carrito, se llama al método refrescarLista(), para reflejar los cambios al usuario.

4. Clase Vista:
   Aunque la lógica principal de encapsulamiento se centró en Articulo, la clase Vista también aplica este concepto. Todos los componentes de se declararon como private y final. Esto para ocultar los componentes de la ventana a otras partes del programa y solo puedan ser inicializados una vez. El acceso a estos atributos es unicamente a través de métodos getters, impidiendo que las demás clases puedan modificar directamente los componentes de la interfaz, respetando el principio de encapsulamiento que buscamos implementar.

4. Resultados

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

public class Articulo {
    private String nombre;
    private double precio;

    public Articulo(String nombre, double precio) {
        setNombre(nombre);
        setPrecio(precio);
    }
```

Figura 1: Inicialización de un artículo (setters y getters)

```java
package mx.unam.fi.poo.p56;
import java.util.ArrayList;
import java.util.List;

public class Carrito {
    private final List<Articulo> articulos; //Este atributo tiene que ser privado y final

    public Carrito() {
        this.articulos = new ArrayList<>(); //Polimorismo
    }
```

Figura 2: Modificaciones de acceso en Carrito

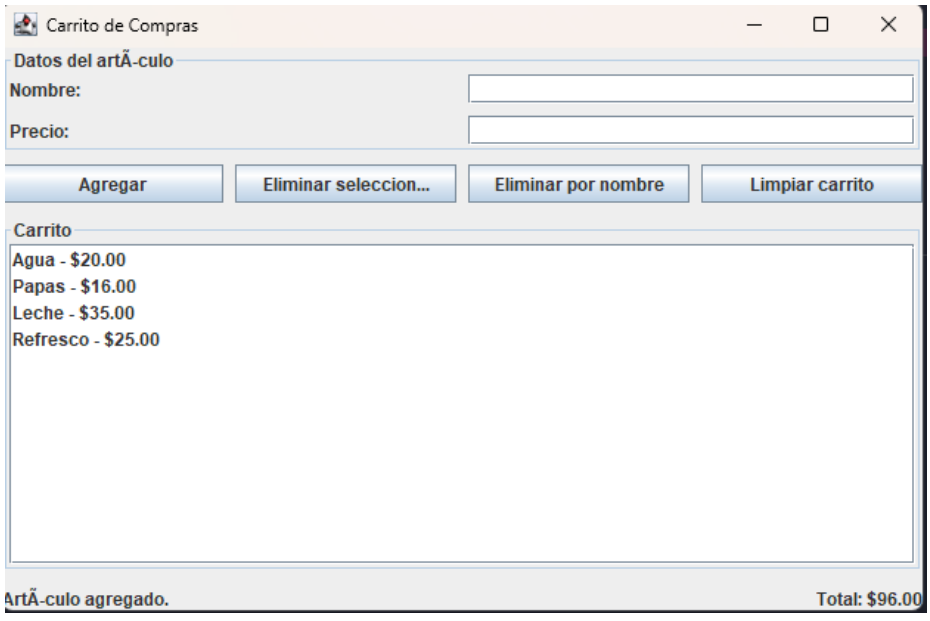

Figura 3: Ejecución: Añadiendo elementos

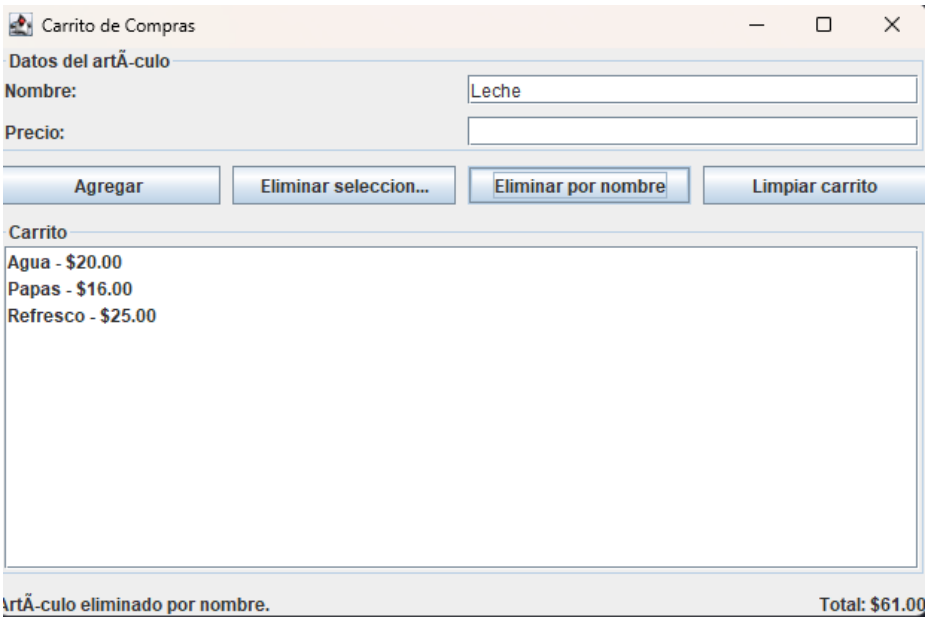

Figura 4: Ejecución: Eliminando por nombre

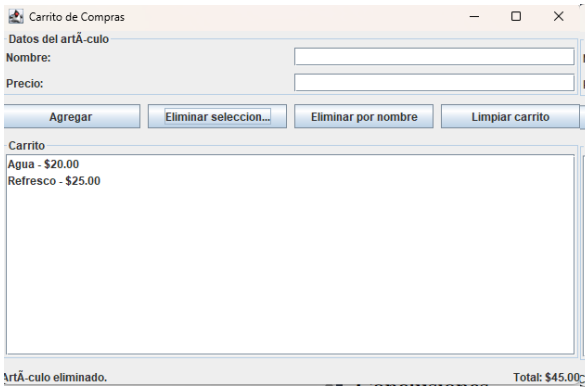

Figura 5: Eliminación por selección

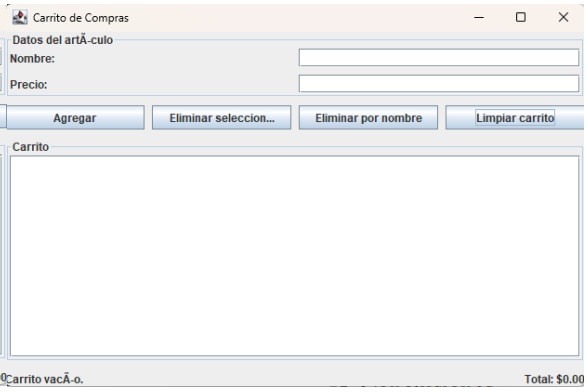

Figura 6: Ejecución: carrito vacío

5. Conclusiones

Los objetivos de la práctica fueron cumplidos. El desarrollo y los resultados obtenidos confirman que la aplicación de los conceptos teóricos fue clave para el éxito del proyecto. Se aplicó correctamente el encapsulamiento para proteger la integridad de los datos en el modelo (Articulo) y para controlar el acceso a los componentes de la interfaz (Vista), lo que resultó en un programa más seguro. La organización de las clases en un paquete demostró ser una práctica muy importante para la estructuración del proyecto. La conexión entre la vista y el modelo funcionó como se esperaba, demostrando una correcta implementación del manejo de eventos. Los principios de la POO aplicados permitieron resolver el problema de forma eficiente y organizada.

6. Referencias

[1] Bailón J. Avila, J. Clases y objetos. en análisis y diseño en poo. (https://portalacademico.cch.unam.mx/cibernetica1/analisis-y-diseno-en-poo/clases-y-objetos), 2022. Consultado el 30-09-2025.

[2] Alianza BUNAM. Conceptos básicos de programación orientada a objetos. (https://alianza.bunam.unam.mx/cch/conceptos-basicos-de-programacion-orientada-a-objetos/), 2022. Consultado el 30-09-2025.

[3] Oracle Java Documentation. Class attributes. (https://docs.oracle.com/en/java/javase/22/docs/api/java.base/java/lang/classfile/Attributes.htm Consultado el 30-09-2025.

[4] Nascor. Introducción a los métodos en java: Tipos y funcionamiento. (https://cursosnascor.com/blog-detalle/introduccion-los-metodos-en-java-tipos-y-funcionamiento), 2023. Consultado el 30-09-2025.

[5] Oracle. Controlling access to members of a class. (https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html). Consultado el 30-09-2025.

[6] Oracle. Creating and using packages. (https://docs.oracle.com/javase/tutorial/java/package/packag Consultado el 30-09-2025.

[7] Oracle. Class arraylist. (https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html), 2025. Consultado el 30-09-2025.

[8] UNAM Portal del CCH. Glosario de cibernetica. (https://portalacademico.cch.unam.mx/glosario/cibernetica). Consultado el 30-09-2025.

[9] J. A. Solano. Clases y objetos. (https://repositorio-uapa.cuaed.unam.mx/repositorio/moodle/pluginfile.php/3068/$\mathrm{mod}_{r}esource/content/1/UAPA-Clases-Objetos/index.html$), 2020. $Consultadoel30-09-2025$.