PROYECTO FINAL

Este archivo se realiza con el próposito de hablar tanto del proyecto como el ejercicio de poo, explicando detalles técnicos para copmprender mejor lo realizado en cada uno

PROYECTO

1. Descripción general del proyecto

Este proyecto lo desarrollamos con el objetivo de crear un sistema de inventario que se pudiera ejecutar desde consola, enfocado en ofrecer las funciones básicas que tendría cualquier sistema comercial: registrar, buscar, actualizar, eliminar, listar productos y calcular el valor total del inventario. Desde el inicio quisimos que fuera un programa claro, con una interfaz sencilla, pero que aplicara correctamente los conceptos de estructuras de datos, condicionales, ciclos, vectores y también el uso de mapas.

2. Estructuras de datos utilizadas

Decidimos trabajar con dos tipos de estructuras para tener una mejor organización de los datos. Por un lado, usamos vectores paralelos para almacenar la información de cada producto: código, nombre, cantidad y precio. Esta estructura nos permitió mantener una lógica secuencial clara y sencilla para manipular los datos. Por otro lado, implementamos un HashMap, que usamos como una especie de índice para asociar los códigos con objetos del tipo Producto. Esto nos facilitó tareas como la búsqueda, actualización y eliminación de productos, ya que el mapa permite acceder directamente a los datos sin necesidad de recorrer todos los vectores.

3. Clase Producto

También creamos una clase llamada Producto, que se encarga de representar cada artículo del inventario. En esa clase almacenamos los atributos principales y definimos algunos métodos básicos para modificar la información cuando sea necesario. Además, implementamos el método toString para que al mostrar los datos en consola se vean organizados y alineados. Esto nos ayudó bastante a mantener una presentación limpia y profesional del inventario cada vez que se imprime.

4. Lógica del menú y control de flujo

Toda la funcionalidad del sistema gira en torno a un menú principal que se repite mientras el usuario no decida salir. Utilizamos una estructura switch para manejar cada opción del menú y un bucle do-while para que el programa permanezca activo hasta que se seleccione la opción de salir. Nos aseguramos de que después de cada operación, el sistema pidiera al usuario presionar ENTER para que pudiera leer con calma los resultados antes de continuar. Este enfoque le da al programa una dinámica clara y controlada.

5. Validaciones y control de errores

También fuimos cuidadosos con las validaciones. Por ejemplo, no permitimos registrar productos con códigos duplicados, ni dejamos que el sistema intente buscar o eliminar productos que no existen. Además, verificamos que no se pueda calcular el valor del inventario si no hay productos registrados. Estas validaciones

fueron importantes para evitar errores y asegurar que el usuario tenga una experiencia más confiable al usar el sistema.

6. Integración de estructuras

Una de las partes más interesantes del desarrollo fue lograr que los vectores y el HashMap trabajaran de forma conjunta sin generar conflictos. Siempre que registramos, actualizamos o eliminamos un producto, nos aseguramos de hacerlo tanto en los arreglos como en el map. Esto nos permitió aprovechar lo mejor de ambas estructuras: la rapidez del HashMap y la simplicidad de los vectores. El resultado fue un sistema que funciona de manera estable y coherente, incluso cuando se realizan múltiples operaciones.

EJERCICIO POO

1. Clase reserva

Esta clase abstracta define la base para todas las reservas. Sus atributos clave son el nombre del cliente, el número de personas y la cantidad de noches. Lo más importante es el método abstracto costo(), que fuerza a cualquier tipo de reserva a implementar su propia lógica para calcular el precio. Esto asegura que cada reserva sepa cómo calcular su propio costo.

2. Clases ReservaNormal y ReservaPremium

Estas dos clases heredan de Reserva y representan tipos específicos de reservas, mostrando un claro ejemplo de polimorfismo.

-ReservaNormal maneja reservas estándar, añadiendo un valor base por persona y noche (valReservaNormal).

-ReservaPremium está diseñada para reservas con servicios adicionales, incluyendo un valor base (valReservaPremium) y un costoExtra fijo.

Ambas clases implementan de forma polimórfica el método costo(), calculando el precio según sus propias reglas. También personalizan el método mostrarInfo() para presentar sus detalles específicos.

3. Clase Hotel

La clase Hotel gestiona la información de un hotel y su reserva asociada. Su atributo principal es una instancia de Reserva, lo que permite que un mismo objeto Hotel pueda manejar cualquier tipo de reserva (normal o premium) sin importar su tipo exacto. Esto es posible gracias al polimorfismo. Los métodos asignarReserva() y mostrarReserva() demuestran cómo el Hotel interactúa de manera genérica con cualquier Reserva, adaptándose automáticamente a su comportamiento específico.

4. Clase Main (Ejecución del Sistema)

La clase Main es el punto de inicio de la aplicación y demuestra la interacción entre todas las clases. Aquí se crean diferentes tipos de reservas (ReservaNormal y ReservaPremium) y se asignan dinámicamente al Hotel. Se observa cómo el polimorfismo permite que el Hotel trate a todas las reservas de manera uniforme a través de la interfaz Reserva, mientras que el comportamiento real de cálculo de costos o visualización de información se adapta automáticamente en tiempo de ejecución al tipo de reserva específica.