1. Pequeñas Clases con Responsabilidad Única

Cada una de nuestras clases está diseñada para manejar únicamente la lógica relacionada con su respectiva entidad. Por ejemplo, la clase "Cliente" se encarga exclusivamente de gestionar la información y operaciones relacionadas con los clientes, como la validación del DNI y la actualización de la dirección. No hemos mezclado funcionalidades de otras entidades dentro de esta clase, lo que ha mantenido nuestras clases pequeñas y manejables.

2. Nombres Claros y Descriptivos

Hemos utilizado nombres claros y descriptivos para todas nuestras clases y métodos. La clase "Producto" se centra en la gestión de productos, mientras que métodos como "CalcularTotalVenta()" en la clase “Venta” reflejan exactamente lo que hacen. Esto nos ha permitido mantener el código legible y fácil de entender, tanto para nosotros como para cualquier otro desarrollador que necesite trabajar en el proyecto.

3. Cohesión

Nos hemos asegurado de que cada clase esté altamente cohesionada. Por ejemplo, en la clase "Producto", todos los métodos y atributos están estrechamente relacionados con la gestión de productos: manejo del stock, precios, y relación con proveedores. No hemos incluido en esta clase lógica que pertenezca a otras áreas del sistema, como ventas o clientes, lo que ha permitido que cada clase haga bien su trabajo específico.

4. Encapsulación

Hemos aplicado la encapsulación de manera efectiva, protegiendo los datos internos de nuestras clases. Los atributos como "dni", "nombre", y `"telefono" en la clase "Cliente" son privados, y solo se pueden acceder a través de métodos públicos controlados, como "GetNombre()" o "SetTelefono()". Esto nos ha ayudado a mantener el control sobre cómo se modifican los datos dentro de nuestras clases.

5. Menor Acoplamiento

Hemos minimizado el acoplamiento entre nuestras clases, asegurándonos de que no dependan fuertemente unas de otras. Por ejemplo, la clase "Ventas" interactúa con "Cliente" y "Empleado", pero esta interacción es mínima y bien definida a través de interfaces o métodos específicos. Esto ha hecho que nuestro sistema sea más flexible, permitiéndonos hacer cambios en una clase sin afectar a otras.

6. Organización de Clases

1. Constantes estáticas públicas: Si nuestra clase contiene constantes, estas se colocan al principio de la clase. Por ejemplo, si en la clase Producto tuviéramos una constante para el valor del IVA, la declararíamos como una constante estática pública al inicio de la clase.

2. Variables estáticas privadas: Después de las constantes públicas, declaramos las variables estáticas privadas, si las hubiera. Estas variables la solemos usar para almacenar datos que son comunes a todas las instancias de la clase, pero no deben ser accesibles desde fuera de la clase.

3. Variables de instancia privadas: Luego, colocamos las variables de instancia privadas, que almacenan el estado de la clase. Estas son las propiedades como nombre, precio, stock, etc., en la clase Producto.

4. Funciones públicas: Después de declarar las variables, colocamos las funciones públicas, que son los métodos que proporcionan la funcionalidad principal de la clase.

5. Funciones privadas: Finalmente, las funciones privadas, que son utilidades internas usadas por las funciones públicas, siguen después de las funciones públicas.