Ejercicio:

Iniciar el proyecto creando la estructura base del sistema de gestión de productos de ventas. Define el propósito de ventas. Define el propósito y los requerimientos básicos.

Lista de objetos:

* Venta.
* Producto.
* Comprador.
* Empleado.
* Orden.
* Tienda.
* Marca.
* Pago.
* Persona.

Lista de atributos y funciones según los objetos:

|  |  |  |
| --- | --- | --- |
| Objetos | Atributos | Métodos |
| Producto | * Cantidad. * Precio. * Nombre. * Código. * Fecha de caducidad * Estado. * Identificador del producto. | * En existencia. * Esta vencido. * Fue devuelto. |
| Tipo de producto | * Nombre de tipo de producto. * Identificador del tipo de producto. | * Mostrar tipo de producto. |
| Comprador | * Forma de pago. * Identificador del comprador | * Comprar. * Realizar pago |
| Empleado | * Identificador de Empleado. * Cargo del Empleado. | * Vender. * Solicitar forma de pago |
| Persona | * Nombre de la persona. * Apellido de la persona. * Edad de la persona. * Identificación de la persona. | * Hablar. * Caminar. * Mostrar información. |
| Tienda | * Nombre de la tienda. * Ubicación de la tienda. * Identificación de la tienda. | * Mostrar nombre de la tienda |
| Marca | * Nombre de la marca. * Identificación de la marca. | * Mostrar nombre de la marca |
| Medio de Pago | * Nombre de tipo de pago. * Identificación de tipo de pago |  |
| Venta | * Identificador de la venta. * Sub total de la venta. * Total de la venta. | Calcular subtotal de la venta.  Calcular total de la venta.  Calcular iva. |
| Orden | * Identificador de la Orden. | * Autorizar compra. * Rechazar compra. |

Definir una clase "Producto" con atributos como nombre, precio y cantidad, e instanciar objetos de tipo producto.

Encapsular los atributos de la clase “Producto” y definir métodos de acceso y mutación para modificar su estado.

Crear una clase base “Persona” e implementar clases derivadas “Cliente” y “Empleado” que hereden sus atributos y métodos.

Sobrescribir el método “Mostrar Información” en las clases derivadas para personalizar la salida según el tipo de objeto (Cliente o Empleado).

Definir una interfaz “IVenta” con el método “CalcularTotal” e implementarla en una clase “Orden” que gestione las ventas del sistema.

Refactorizar el código del sistema para cumplir con los principios SOLID. Documentar el proyecto final.

OrderService.cs

public class OrderService : IOrder

{

    GetBoolDataService getBoolDataService = new GetBoolDataService();

    SaleService saleService = new SaleService();

    public string ValueEntry { get; set; }

    public string CanContinue()

    {

        ValueEntry = getBoolDataService.GetData();

        do

        {

            SubSaleService subSaleService = new SubSaleService();

            subSaleService.CalculateSubTotal();

            ValueEntry = getBoolDataService.GetData();

        } while (ValueEntry != "false");

        saleService.CalculateTotal();

        return ValueEntry;

    }

}

SubSaleService.cs

public class SubSaleService : ISubSale

{

    GetNumericDataService getNumericDataService = new GetNumericDataService();

    GetStringDataService getStringDataService = new GetStringDataService();

    public decimal CalculateSubTotal()

    {

        Product product = new Product();

        Console.WriteLine("Enter product name: ");

        product.ProductName = getStringDataService.GetData();

        Console.WriteLine("Enter product description: ");

        product.ProductDescription = getStringDataService.GetData();

        Console.WriteLine("Enter product amount: ");

        product.ProductAmount = int.Parse(getNumericDataService.GetData());

        Console.WriteLine("Enter product price: ");

        product.ProductPrice = decimal.Parse(getNumericDataService.GetData());

        SubSale subSale = new SubSale();

        subSale.SubTotal = product.ProductAmount \* product.ProductPrice;

        Console.WriteLine($"Product name: {product.ProductName}");

        Console.WriteLine($"Product description: {product.ProductDescription}");

        Console.WriteLine($"Product amount: {product.ProductAmount} ");

        Console.WriteLine($"Product price: {product.ProductPrice} ");

        Console.WriteLine($"Subtotal: {subSale.SubTotal}"

        );

        return subSale.SubTotal;

    }

}

public class SaleService : ISale

{

    List<SubSale> SubSales { get; set; } = new List<SubSale>();

    public decimal CalculateTotal()

    {

        Console.WriteLine("Calculating total sale...");

        foreach (var subSale in SubSales)

        {

            Console.WriteLine($"Subtotal: {subSale.SubTotal}");

        }

        decimal totalSale = SubSales.Sum(subSale => subSale.SubTotal);

        return totalSale;

    }

}

Program.cs

internal class Program { private static void Main(string[] args) { OrderService orderService = new OrderService(); orderService.CanContinue(); } }

5 principios de mejora del software:

SOLID:

**SRP**: principio de responsabilidad única, cada clase tiene una única responsabilidad y solo una razón para cambiar. (hacer solo una funcionalidad).

**OCP**: Principio de apertura/cerrado, las entidades de software, deben estar abiertas pero cerradas a la modificación.

**LSP**: Principio de sustitución de Liskov, Los objetos de una superclase deben ser reemplazables por objetos de su subclase sin afectar la corrección del programa.

**ISP**: Principio de segregación de interfaz, Los clientes no deberían verse obligados a depender de interfaces que no utilizan.

**DIP**: Principio de inversión de dependen, Los módulos de alto nivel no deberían depender de módulos de bajo nivel. Ambos deberían depender de abstracciones.