Лабораторна робота №2

Тема: Принципи програмування. SOLID

Мета роботи: навчитися дотримуватися принципів SOLID та обґрунтовувати їх. Навчитися описувати дизайн програми за допомогою UML діаграм

Хід роботи:

Завдання на лабораторну роботу:

1. ***Завдання:***

Завдання 0: Підготовка до виконання завдання

1. Встановити DotNet CLI 6 або 7 версії.
2. Створити окремий репозиторій на GitHub або GitLab. В цьому репозиторії будуть міститися всі виконані Вами завдання курсу “Конструювання програмного забезпечення”. Кожне завдання буде міститися в окремій директорії **`lab-1`**, **`lab-2`** ... **`lab-n`**
3. Надати доступ викладачу *mykola-fant-ztu <kipz_fmo@ztu.edu.ua>*
4. Склонувати створений репозиторій.
5. Створити директорію **`lab-1`**
6. Зайти в створену директорію і запустити команду **`dotnet new console`**
7. Перейти до Завдання 1 🙂

Завдання 1: Виконати завдання з дотриманням принципів SOLID.

1. Запрограмуйте клас ***Money*** (об'єкт класу оперує однією валютою) для роботи з грошима. У класі мають бути передбачені: поле для зберігання цілої частини грошей (долари, євро, гривні тощо) і поле для зберігання копійок (центи, євроценти, копійки тощо). Реалізувати методи виведення суми на екран, задання значень частин.
2. Створити клас ***Product*** для роботи з продуктом або товаром. Реалізувати метод, який дозволяє зменшити ціну на задане число.
3. Реалізувати клас ***Warehouse***, який описує товари, що зберігаються на складі: найменування, одиниця виміру, ціна одиниці, кількість, дата останнього завозу, тощо.
4. Реалізувати клас ***Reporting*** для роботи зі звітністю. Реєстрація надходження товару (формування прибуткової

накладної) і відвантаження (видаткова накладна). Звіт по інвентаризації (залишки на складі).

5. Для кожного з класів реалізувати необхідні методи і поля. Для класів передбачити реалізацію конструкторів та методів для встановлення та читання значень.
6. Ви також можете додавати власний функціонал для унаочнення принципів SOLID. Приклади додаткового функціоналу:
   a. категорії для продуктів;
   b. конкретні дочірні класи валюти
   c. корзина для замовлень.

Завдання 2: Написати код для тестування отриманої функціональності.

1. Покажіть правильність роботи свого коду запустивши його в головному методі програми.

Завдання 3: Опишіть особливості дотримання принципів SOLID в Вашому коді

1. За допомогою коментарів додайте пояснення і обґрунтуваня кожного з п'яти принципів SOLID біля відповідних рядків коду.
2. Коментарі можна залишати українською або (бажано) англійською мовами.

Завдання 4: UML діаграма

1. Підготувати діаграму створених у програмі класів та інтерфейсів за допомогою https://app.diagrams.net/.
2. Експортувати створену діаграму та запушити експортований файл у кореневу директорію проекту.

2. *Лістинг програми:*

a. *Бібліотека класів:*

i. *Money.cs:*

```
using System.Globalization;

namespace SOLIDLibrary
{
    public class Money
    {
        private int _bills;
        private int _coins;
        private char _currency;

        public int Bills
        {
            get => this._bills;
            set
```

```
            {
                if (value >= 0)
                    this._bills = value;
                else
                    this._bills = 0;
            }
        }

        public int Coins
        {
            get => this._coins;
            set
            {
                if ((value >= 0) && (value <= 99))
                    this._coins = value;
                else
                    this._coins = 0;
            }
        }

        public char Currency
        {
            get => this._currency;
            set
            {
                if (CharUnicodeInfo.GetUnicodeCategory(value) ==
UnicodeCategory.CurrencySymbol)
                    this._currency = value;
                else
                    this._currency = '$';
            }
        }

        public Money(int bills = 0, int coins = 0, char currency = '$')
        {
            this.Bills = bills;
            this.Coins = coins;
            this.Currency = currency;
        }

        // Single Responsibility Principle - method does one certain thing
        // Also DRY - Don't repeat yourself. Method used 3 times
        public double GetMoneyNumber() => (double)(this.Bills + this.Coins / 100.0);

        // Open-Closed Principle - open to expansion, closed to changes
        public override string ToString() => String.Format("{0}{1}", this.Currency,
this.GetMoneyNumber());
    }
}
```

ii. *Product.cs:*

```
namespace SOLIDLibrary
{
    public class Product : Money
    {
        private int _quantity;
        private string _dimension;

        public string Name { get; set; }
        public string Description { get; set; }
        public int Quantity
        {
            get => this._quantity;
            set
            {
```

```
                if (value >= 0)
                    this._quantity = value;
                else
                    this._quantity = 0;
            }
        }
        public string Dimension
        {
            get => this._dimension;
            set
            {
                if (value.Trim() == "piece" || value.Trim() == "kilo" || value.Trim() ==
"шт." || value.Trim() == "кг.")
                    this._dimension = value;
                else
                    this._dimension = "piece";
            }
        }

        public Product(string name, string description, int quantity, string dimension,
int bills, int coins, char currency) : base(bills, coins, currency)
        {
            this.Name = name;
            this.Description = description;
            this.Quantity = quantity;
            this.Dimension = dimension;
        }

        // Single Responsibility Principle
        public void ReducePrice(double ReducePrice)
        {
            if (this.GetMoneyNumber() - ReducePrice > 0)
            {
                int ReduceBills = (int)ReducePrice;
                int ReduceCoins = (int)(Math.Round(ReducePrice % 1, 2) * 100);
                this.Bills -= ReduceBills;
                this.Coins -= ReduceCoins;
                return;
            }

            Console.WriteLine($"Помилка: Не можна зменшити на ціну: {ReducePrice}");
        }

        // Open-Closed Principle
        public override string ToString()
        {
            return $"\n\tПро продукт\n\nНазва: {this.Name}\nОпис: {this.Description}\nКі-
лькість: {this.Quantity}{this.Dimension}\nЦіна: {this.GetMoneyNumber()}";
        }
    }
}
```

iii. Warehouse.cs:

```
namespace SOLIDLibrary
{
    public class Warehouse
    {
        private double _area;

        public List<Product> Products { get; set; }
        public string Name;
        public string Adress;
        public double Area
        {
            get => this._area;
```

```
            set
            {
                if (value > 0)
                    this._area = value;
                else
                    this._area = 0;
            }
        }
        public DateTime LastDelivery;

        public Warehouse(string name, string adress, double area)
        {
            this.Products = new List<Product>();
            this.Name = name;
            this.Adress = adress;
            this.Area = area;
        }

        // Single Responsibility Principle
        public void AddProduct(Product NewProduct)
        {
            if (!this.Products.Contains(NewProduct))
            {
                this.Products.Add(NewProduct);
                this.LastDelivery = DateTime.Now;
            }
        }

        // Single Responsibility Principle
        public void RemoveProduct(Product ExistingProduct)
        {
            if (this.Products.Contains(ExistingProduct))
                this.Products.Remove(ExistingProduct);
        }

        // Open-Closed Principle
        public string GetProductsAsString()
        {
            string OutputString = "";

            for (int i = 0; i < this.Products.Count; i++)
            {
                OutputString += this.Products[i].ToString();
            }

            return OutputString;
        }

        // Open-Closed Principle
        public override string ToString()
        {
            string ProductsString = GetProductsAsString();
            if (ProductsString == "")
                ProductsString = "Товарів на складі немає!";

            return $"\n\tПро склад:\n\nНазва: {this.Name}\nАдреса: {this.Adress}\nПлоща:
{this.Area}\n{ProductsString}";
        }
    }
}
```

iv. *Reporting.cs:*

```
namespace SOLIDLibrary
{
    // Interface Segregation Principle
```

```
    interface IInvoiceGenerator
    {
        void GenerateInvoice(Product SomeProduct, int Quantity);
    }

    interface IInventoryManager
    {
        void UpdateInventory(Product SomeProduct, int Quantity);
        Dictionary<Product, int> Inventory { get; }
    }

    class RevenueGenerator : IInvoiceGenerator
    {
        public void GenerateInvoice(Product SomeProduct, int Quantity)
        {
            Console.WriteLine($"Отримано {Quantity} {SomeProduct.Name}. Формування прибу-
ткової накладної...\n");
        }
    }

    class ExpenditureGenerator : IInvoiceGenerator
    {
        public void GenerateInvoice(Product SomeProduct, int Quantity)
        {
            Console.WriteLine($"Прибуло {Quantity} {SomeProduct.Name}. Формування видат-
кової накладної...\n");
        }
    }

    public class InventoryManager : IInventoryManager
    {
        private Dictionary<Product, int> _inventory = new Dictionary<Product, int>();

        public Dictionary<Product, int> Inventory
        {
            get => _inventory;
        }

        public void UpdateInventory(Product NewProduct, int Quantity)
        {
            if (Inventory.ContainsKey(NewProduct))
                Inventory[NewProduct] += Quantity;
            else
                Inventory.Add(NewProduct, Quantity);
        }

        public void GenerateInventoryReport()
        {
            Console.WriteLine("Проводимо інвентаризацію...");

            if (Inventory.Count > 0)
            {
                Console.WriteLine("Звіт про інвентаризацію:");

                foreach (KeyValuePair<Product, int> product in Inventory)
                    Console.WriteLine($"{product.Key.Name}: {product.Value}");

                return;
            }

            Console.WriteLine("Продуктів на складі немає.\n");
        }
    }

    public class Reporting
```

```
    {
        private RevenueGenerator _revenueGenerator;
        private ExpenditureGenerator _expenditureGenerator;
        private InventoryManager _inventoryManager;

        public Reporting()
        {
            this._revenueGenerator = new RevenueGenerator();
            this._expenditureGenerator = new ExpenditureGenerator();
            this._inventoryManager = new InventoryManager();
        }

        // Single Responsibility Principle
        public void RegisterReceipt(Product SomeProduct, int Quantity)
        {
            _inventoryManager.UpdateInventory(SomeProduct, Quantity);
            _revenueGenerator.GenerateInvoice(SomeProduct, Quantity);
        }

        // Single Responsibility Principle
        public void RegisterShipment(Product SomeProduct, int Quantity)
        {
            if (_inventoryManager.Inventory.ContainsKey(SomeProduct))
            {
                int availableQuantity = _inventoryManager.Inventory[SomeProduct];
                if (availableQuantity >= Quantity)
                {
                    _inventoryManager.UpdateInventory(SomeProduct, -Quantity);
                    _expenditureGenerator.GenerateInvoice(SomeProduct, Quantity);
                }
                else
                    Console.WriteLine($"Помилка: Не достатньо {SomeProduct} для доставки
{Quantity} кількості.\n");
            }
            else
                Console.WriteLine($"Помилка: {SomeProduct} немає в наявності.\n");
        }

        // Open-Closed Principle
        public void GenerateInventoryReport()
        {
            _inventoryManager.GenerateInventoryReport();
        }
    }
}
```

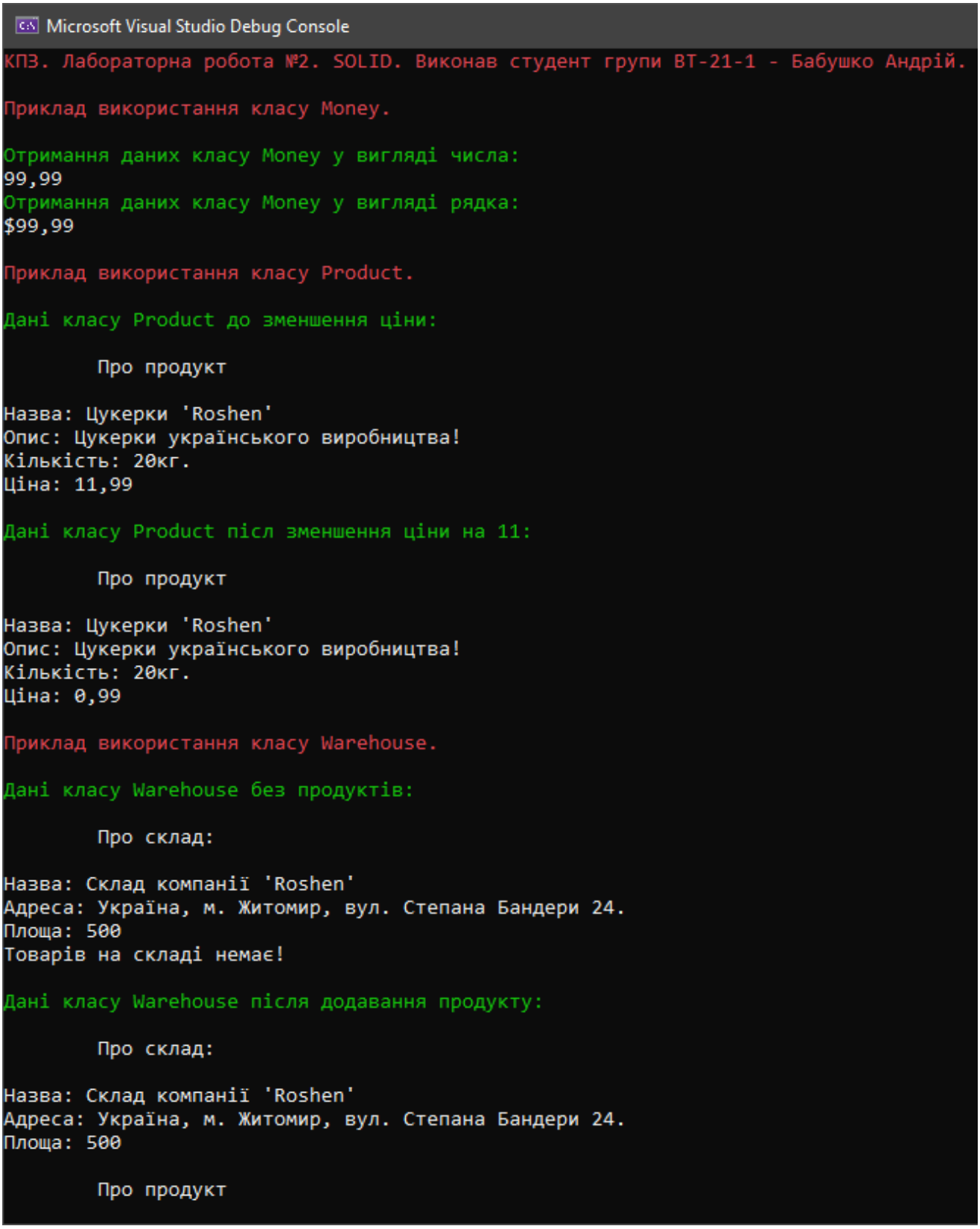
b. *Консольний додаток:*

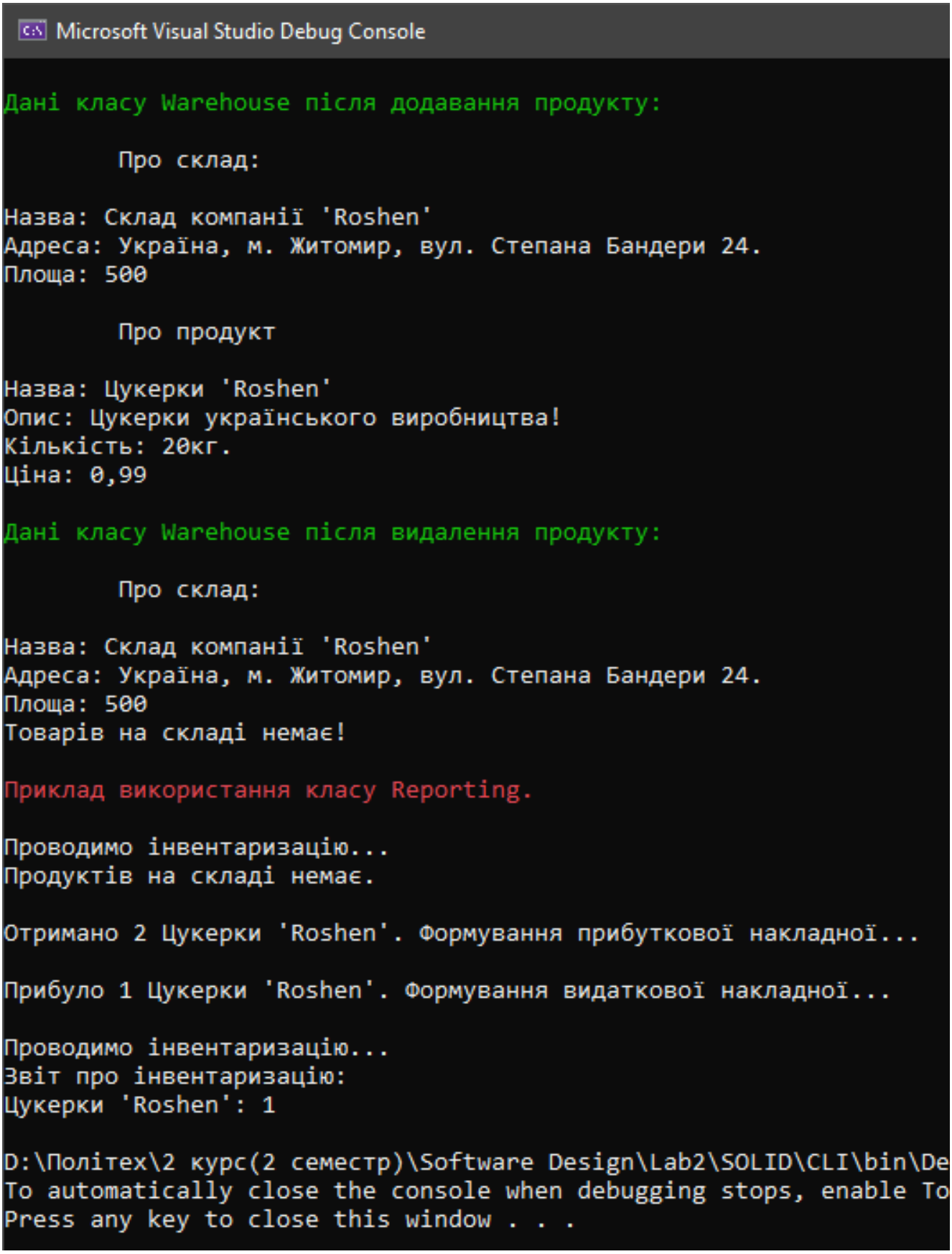
i. *Program.cs:*

```
using SOLIDLibrary;
using System.Text;

Console.InputEncoding = Encoding.UTF8;
Console.OutputEncoding = Encoding.UTF8;

void BeautifyOutput(string Output, string OutputColor)
{
    if(OutputColor == "Green")
        Console.ForegroundColor = ConsoleColor.Green;

    if (OutputColor == "Red")
        Console.ForegroundColor = ConsoleColor.Red;

    Console.WriteLine(Output);
    Console.ForegroundColor = ConsoleColor.White;
}
```

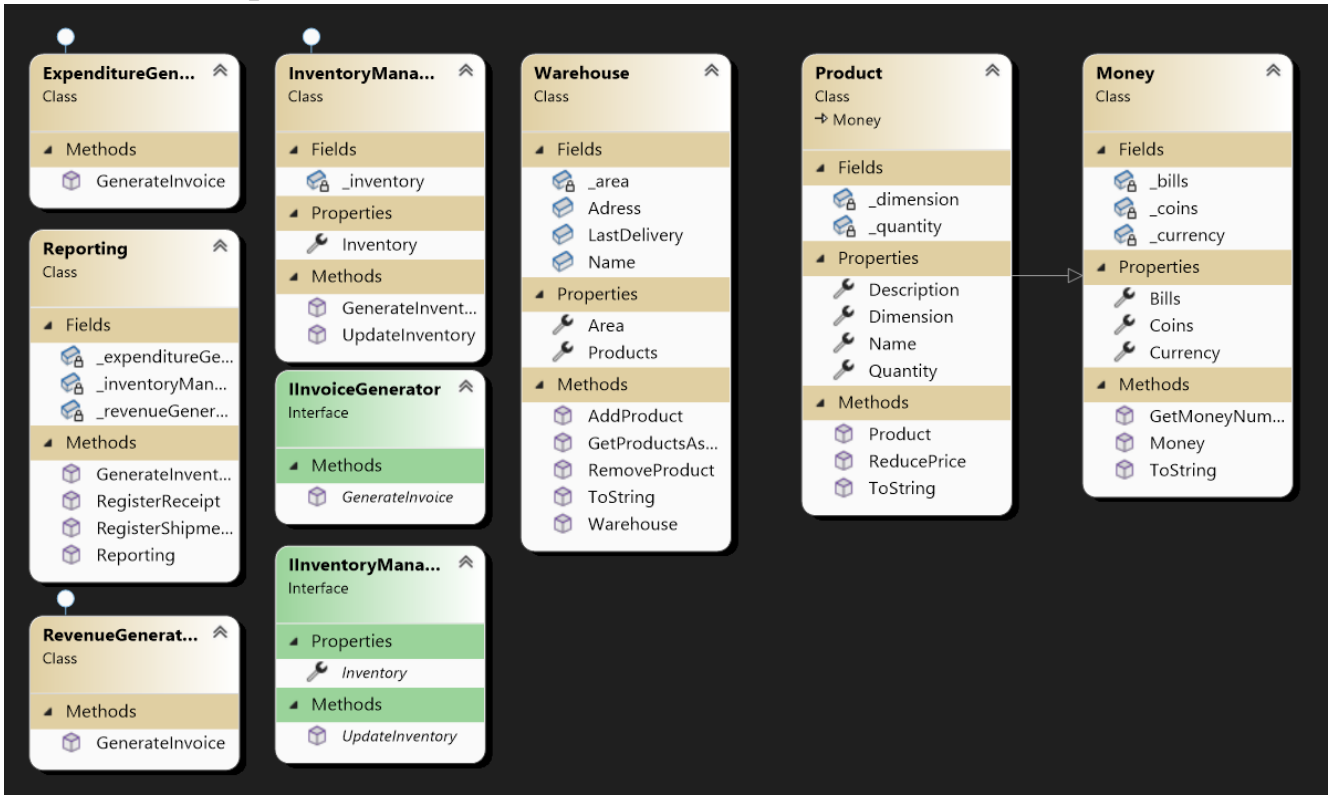
```
BeautifyOutput("КПЗ. Лабораторна робота №2. SOLID. Виконав студент групи ВТ-21-1 - Бабушко Андрій.", "Red");

BeautifyOutput("\nПриклад використання класу Money.\n", "Red");
Money MyMoney = new Money(99, 99, '$');
BeautifyOutput("Отримання даних класу Money у вигляді числа:", "Green");
Console.WriteLine(MyMoney.GetMoneyNumber());
BeautifyOutput("Отримання даних класу Money у вигляді рядка:", "Green");
Console.WriteLine(MyMoney);

BeautifyOutput("\nПриклад використання класу Product.", "Red");
Product MyProduct = new Product("Цукерки 'Roshen'", "Цукерки українського виробництва!",
20, "кг.", 11, 99, '€');
BeautifyOutput("\nДані класу Product до зменшення ціни:", "Green");
Console.WriteLine(MyProduct);
MyProduct.ReducePrice(11);
BeautifyOutput("\nДані класу Product після зменшення ціни на 11:", "Green");
Console.WriteLine(MyProduct);

BeautifyOutput("\nПриклад використання класу Warehouse.", "Red");
Warehouse MyWarehouse = new Warehouse("Склад компанії 'Roshen'", "Україна, м. Житомир,
вул. Степана Бандери 24.", 500);
BeautifyOutput("\nДані класу Warehouse без продуктів:", "Green");
Console.WriteLine(MyWarehouse);
MyWarehouse.AddProduct(MyProduct);
BeautifyOutput($"\nДані класу Warehouse після додавання продукту:", "Green");
Console.WriteLine(MyWarehouse);
MyWarehouse.RemoveProduct(MyProduct);
BeautifyOutput($"\nДані класу Warehouse після видалення продукту:", "Green");
Console.WriteLine(MyWarehouse);

BeautifyOutput("\nПриклад використання класу Reporting.\n", "Red");
Reporting MyReporting = new Reporting();
MyReporting.GenerateInventoryReport();
MyReporting.RegisterReceipt(MyProduct, 2);
MyReporting.RegisterShipment(MyProduct, 1);
MyReporting.GenerateInventoryReport();
```

3. *Результати програми:*


Microsoft Visual Studio Debug Console
КПЗ. Лабораторна робота №2. SOLID. Виконав студент групи ВТ-21-1 - Бабушко Андрій.
Приклад використання класу Money.
Отримання даних класу Money у вигляді числа:
99,99
Отримання даних класу Money у вигляді рядка:
$99,99
Приклад використання класу Product.
Дані класу Product до зменшення ціни:
Про продукт
Назва: Цукерки 'Roshen'
Опис: Цукерки українського виробництва!
Кількість: 20кг.
Ціна: 11,99
Дані класу Product післ зменшення ціни на 11:
Про продукт
Назва: Цукерки 'Roshen'
Опис: Цукерки українського виробництва!
Кількість: 20кг.
Ціна: 0,99
Приклад використання класу Warehouse.
Дані класу Warehouse без продуктів:
Про склад:
Назва: Склад компанії 'Roshen'
Адреса: Україна, м. Житомир, вул. Степана Бандери 24.
Площа: 500
Товарів на складі немає!
Дані класу Warehouse після додавання продукту:
Про склад:
Назва: Склад компанії 'Roshen'
Адреса: Україна, м. Житомир, вул. Степана Бандери 24.
Площа: 500
Про продукт


Microsoft Visual Studio Debug Console
Дані класу Warehouse після додавання продукту:
Про склад:
Назва: Склад компанії 'Roshen'
Адреса: Україна, м. Житомир, вул. Степана Бандери 24.
Площа: 500
Про продукт
Назва: Цукерки 'Roshen'
Опис: Цукерки українського виробництва!
Кількість: 20кг.
Ціна: 0,99
Дані класу Warehouse після видалення продукту:
Про склад:
Назва: Склад компанії 'Roshen'
Адреса: Україна, м. Житомир, вул. Степана Бандери 24.
Площа: 500
Товарів на складі немає!
Приклад використання класу Reporting.
Проводимо інвентаризацію...
Продуктів на складі немає.
Отримано 2 Цукерки 'Roshen'. Формування прибуткової накладної...
Прибуло 1 Цукерки 'Roshen'. Формування видаткової накладної...
Проводимо інвентаризацію...
Звіт про інвентаризацію:
Цукерки 'Roshen': 1
D:\Політех\2 курс(2 семестр)\Software Design\Lab2\SOLID\CLI\bin\De
To automatically close the console when debugging stops, enable To
Press any key to close this window . . .

4. *UML діаграма:*



Висновок: під час виконання лабораторної роботи було отримано навички використання принципів SOLID в програмуванні при написанні певних класів з використанням інтерфейсів та принципів ООП.