ЛАБОРАТОРНА РОБОТА № 1

Тема: Принципи програмування. DRY, KISS, SOLID, YAGNI та ін.

Мета: навчитися дотримуватися принципів програмування та обґрунтовувати їх.

Завдання 1 (Варіант 1): Виконати завдання з дотриманням відомих Вам принципів програмування.

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 програмування. Приклади додаткового функціоналу:
   a. категорії для продуктів;
   b. конкретні дочірні класи валюти
   c. корзина для замовлень.

Лістинг програми:

```
using System;

public class Money
{
```

					ДУ «Житомирська політехніка».22.121.18.000 - Лр1		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Пасічник Б.Р			Звіт з лабораторної роботи	Літ.	Арк. 1 / Аркушів ZZ
Перевір.		Фант М.О..					
Керівник						ФІКТ Гр. ВТ-22-1[2]	
Н. контр.							
Зав. каф.							

```
 private int notes;
    private int coins;

    public Money(int notes = 0, int coins = 0)
    {
        this.WholePart = notes;
        this.FractionPart = coins;
    }

    public int WholePart
    {
        get { return notes; }
        set { notes = value; }
    }

    public int FractionPart
    {
        get { return coins; }
        set
        {
            if (value < 0)
            {
                throw new ArgumentOutOfRangeException(nameof(value), "Копійки не мо-
жуть становити від'ємну суму");
            }
            notes += value / 100;
            coins = value % 100;
        }
    }

    public void Display()
    {
        Console.WriteLine($"{notes}.{coins:D2}");
    }

    public void Decrease(int amountNotes, int amountCoins)
    {
        int totalCoins = (notes * 100 + coins) - (amountNotes * 100 + amountCoins);
        if (totalCoins < 0)
        {
            throw new ArgumentOutOfRangeException("Зменшення перевищує поточну су-
му");
        }
        notes = totalCoins / 100;
        coins = totalCoins % 100;
    }

    public void Increase(int amountNotes, int amountCoins)
    {
        int totalCoins = (notes * 100 + coins) + (amountNotes * 100 + amountCoins);
```

```
        notes = totalCoins / 100;
        coins = totalCoins % 100;
    }
}

public class Product
{
    private string name;
    private Money price;

    public Product(string name, int priceNotes, int priceCoins)
    {
        this.Name = name;
        this.Price = new Money(priceNotes, priceCoins);
    }

    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public Money Price
    {
        get { return price; }
        set { price = value; }
    }

    public void DecreasePrice(int amountNotes, int amountCoins)
    {
        price.Decrease(amountNotes, amountCoins);
    }

    public void Display()
    {
        Console.WriteLine($"Назва: {Name}, Ціна: ");
        Price.Display();
    }
}


public class Warehouse
{
    private List<Product> products;
    private string unit;
    private int quantity;
    private DateTime lastDeliveryDate;

    public Warehouse(string unit)
    {
```

```
        this.products = new List<Product>();
        this.unit = unit;
    }

    public string Unit
    {
        get { return unit; }
        set { unit = value; }
    }

    public int Quantity
    {
        get { return quantity; }
        set { quantity = value; }
    }

    public DateTime LastDeliveryDate
    {
        get { return lastDeliveryDate; }
        set { lastDeliveryDate = value; }
    }

    public void AddProduct(Product product, int quantity, DateTime deliveryDate)
    {
        products.Add(product);
        this.quantity += quantity;
        this.lastDeliveryDate = deliveryDate;
    }

    public void RemoveProduct(Product product, int quantity)
    {
        if (this.quantity < quantity)
        {
            throw new InvalidOperationException("Недостатня кількість товару на скла-
ді");
        }
        products.Remove(product);
        this.quantity -= quantity;
    }

    public void Display()
    {
        Console.WriteLine($"Одиниця виміру: {unit}, Кількість: {quantity}, Дата
останнього завозу: {lastDeliveryDate.ToShortDateString()}");
        foreach (var product in products)
        {
            product.Display();
        }
    }
```

```
    public List<Product> GetProducts()
    {
        return products;
    }
}

public class Reporting
{
    private Warehouse warehouse;

    public Reporting(Warehouse warehouse)
    {
        this.warehouse = warehouse;
    }

    public void RegisterIncomingProduct(Product product, int quantity, DateTime
deliveryDate)
    {
        warehouse.AddProduct(product, quantity, deliveryDate);
        Console.WriteLine($"Продукт {product.Name} було завезено в кількості
{quantity} {warehouse.Unit} на склад.");
    }

    public void RegisterOutgoingProduct(Product product, int quantity)
    {
        warehouse.RemoveProduct(product, quantity);
        Console.WriteLine($"Продукт {product.Name} було відвантажено в кількості
{quantity} {warehouse.Unit} зі складу.");
    }

    public void InventoryReport()
    {
        Console.WriteLine("Звіт по інвентаризації:");
        warehouse.Display();
    }
}

// Приклад використання
public class Program
{
    public static void Main()
    {
        // Створення продуктів
        Product laptop = new Product("Laptop", 1500, 50);
        Product phone = new Product("Phone", 800, 25);

        // Створення складу
        Warehouse warehouse = new Warehouse("шт.");

        // Створення об'єкта для звітності
```

```
        Reporting reporting = new Reporting(warehouse);

        // Реєстрація надходження товарів
        reporting.RegisterIncomingProduct(laptop, 10, DateTime.Now);
        reporting.RegisterIncomingProduct(phone, 20, DateTime.Now);

        // Звіт по інвентаризації
        reporting.InventoryReport();

        // Реєстрація відвантаження товарів
        reporting.RegisterOutgoingProduct(phone, 5);

        // Звіт по інвентаризації після відвантаження
        reporting.InventoryReport();
    }
}
```

Завдання 2: Написати код для тестування отриманої функціональності.

1. Покажіть правильність роботи свого коду запустивши його в головному методі програми.
2. Достатньо буде просто вивести певну інформацію, щоб показати, що класи комунікують певним чином між собою.

```
Продукт Laptop було завезено в кількості 10 шт. на склад.
Продукт Phone було завезено в кількості 20 шт. на склад.
Звіт по інвентаризації:
Одиниця виміру: шт., Кількість: 30, Дата останнього завозу: 26.05.2024
Назва: Laptop, Ціна:
1500.50
Назва: Phone, Ціна:
800.25
Продукт Phone було відвантажено в кількості 5 шт. зі складу.
Звіт по інвентаризації:
Одиниця виміру: шт., Кількість: 25, Дата останнього завозу: 26.05.2024
Назва: Laptop, Ціна:
1500.50
```

Завдання 3: Опишіть особливості дотримання принципів програмування в Вашому коді

1. Додайте файл README.md в кореневу директорію цієї лабораторної роботи. В файлі README.md опишіть дотримання окремо кожного принципу програмування, який Вам відомо, і який можна продемонструвати Вашим кодом.
2. Опис можна залишати українською або (бажано) англійською мовами.
3. Ваш опис повинен містити посилання на відповідні файли і рядки коду.
4. Як залишати посилання на свої рядки коду можна глянути тутечки (для посилання на директорію) або тут (для посилання на окремі рядки).
5. Синтаксис .md файлів документації можна знайти туть або туть.
6. Для отримання максимальної оцінки Ви повинні продемонструвати мінімум **7** принципів. SOLID принципи рахуються окремо. Повний список принципів, які було розглянуто на лекції:

1. # Принципи програмування, які були використані у цьому проєкті:

1. DRY (Don't Repeat Yourself)

Принцип DRY спрямований на зменшення повторення коду. Це видно в тому, як клас `Money` робить конвертацію між банкнотами і монетами.
- **Відповідний код**:
  - [Money.cs](./Money.cs) - Рядки [3-11](./Money.cs#L3-L11)

2. KISS (Keep It Simple, Stupid)

Принцип KISS полягає у тому, що системи повинні бути максимально простими. Це видно у класі `Product`, який інкапсулює деталі продукту зрозумілим чином.
- **Відповідний код**: [Product.cs](./Product.cs) - Рядки [3-9](./Product.cs#L3-L9)

3. Принципи SOLID

a. SRP (Single Responsibility Principle)

Кожен клас повинен виконувати лише одну задачу. Клас `Money` обробляє валюту, клас `Product` обробляє деталі продукту, а клас `Warehouse` обробляє інвентар.
- **Відповідний код**:
  - [Money.cs](./Money.cs)
  - [Product.cs](./Product.cs)
  - [Warehouse.cs](./Warehouse.cs)

b. OCP (Open/Closed Principle)

Класи повинні бути відкритими для розширення, але закритими для модифікації. Клас `Money` може бути розширений новими функціями без зміни існуючого коду.
- **Відповідний код**: [Money.cs](./Money.cs)

c. LSP (Liskov Substitution Principle)

Похідні класи повинні бути підстановлюваними замість базових класів. Цей принцип не демонструється безпосередньо у цьому прикладі, бо нема прикладу наслідування, але структура відповідає цьому принципу.

d. ISP (Interface Segregation Principle)

Жоден код не повинен залежати від методів, які він не використовує. Класи `Product` і `Warehouse` не мають зайвих методів, які не відносяться до їх функціональності.
- **Відповідний код**:
  - [Product.cs](./Product.cs)
  - [Warehouse.cs](./Warehouse.cs)

e. DIP (Dependency Inversion Principle)

Суть полягає у розриві зв'язності між програмними модулями вищого та нижчого рівнів за допомогою спільних абстракцій. Клас `Reporting` залежить від інтерфейсу `Warehouse`, що дозволяє легше вносити зміни та розширення.
- **Відповідний код**:
  - [Reporting.cs](./Reporting.cs) - Рядки [3-5](./Reporting.cs#L3-L5)

4. YAGNI (You Aren't Gonna Need It)

Цей принцип стверджує, що функціонал не повинен додаватися до того, як він стане необхідним. Я застосував цей принцип під час написання коду.
- **Відповідний код**:
  - [Money.cs](./Money.cs)

- [Product.cs](./Product.cs)
- [Warehouse.cs](./Warehouse.cs)
- [Reporting.cs](./Reporting.cs)

5. COI (Composition Over Inheritance)

Цей принцип пропонує використовувати складання замість успадкування для досягнення поліморфізму. Клас `Product` використовує клас `Money` через складання, а не через успадкування.
- **Відповідний код**:
  - [Product.cs](./Product.cs) - Рядки [6-8](./Product.cs#L6-L8)

6. Program to Interfaces not Implementations

Цей принцип продемонстровано через клас `Warehouse`, який може бути легко абстрагований до інтерфейсу, що дозволяє іншим класам взаємодіяти з ним, не знаючи його реалізації.
- **Відповідний код**:
  - [Warehouse.cs](./Warehouse.cs)

7. Fail Fast

Клас `Money` кидає винятки одразу при спробі виконання недійсних операцій, наприклад, при встановленні від'ємного значення для монет.
- **Відповідний код**:
  - [Money.cs](./Money.cs) - Рядки [17-20](./Money.cs#L17-L20)

Висновки: під час виконання лабораторної роботи я повторно закріпив принципи програмування. Також я дізнався як вставляти діаграми та дізнався як вставляти посилання на ділянки коду.