***Лабораторна робота №2***

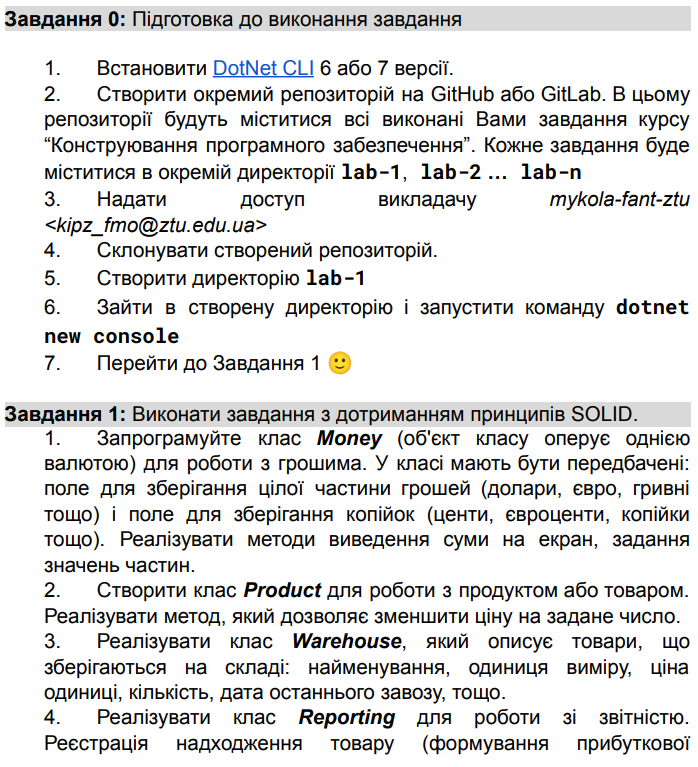
***Тема: Принципи програмування. SOLID***

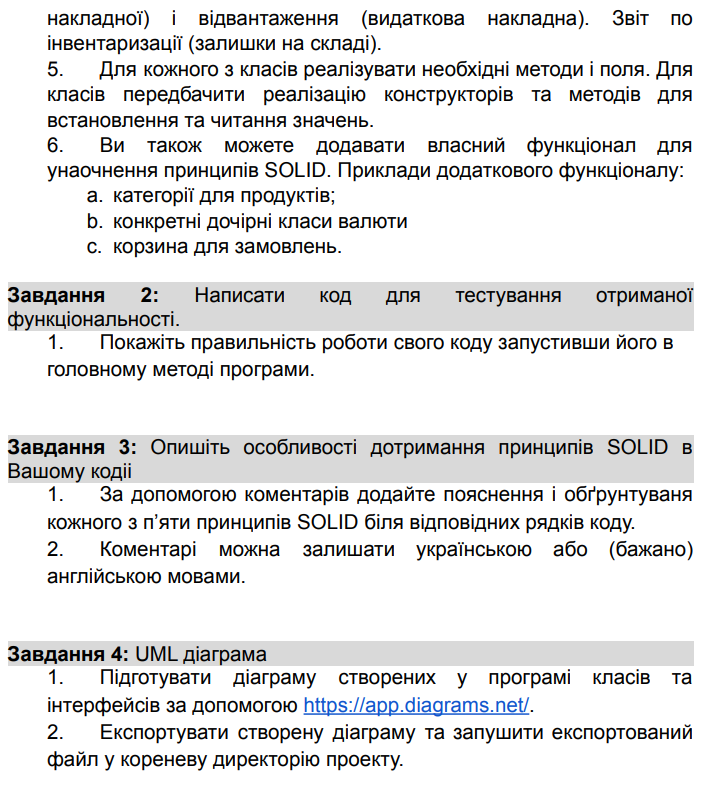
***Мета роботи***: навчитися дотримуватися принципів SOLID та обґрунтовувати їх. Навчитися описувати дизайн програми за допомогою UML діаграм

***Хід роботи:***

***Завдання на лабораторну роботу:***

1. ***Завдання:***

******



1. ***Лістинг програми:***
   1. ***Бібліотека класів:***
      1. ***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());

}

}

* + 1. ***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}\nКількість: {this.Quantity}{this.Dimension}\nЦіна: {this.GetMoneyNumber()}";

}

}

}

* + 1. ***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}";

}

}

}

* + 1. ***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 прибуткової накладної...\n");

}

}

class ExpenditureGenerator : IInvoiceGenerator

{

public void GenerateInvoice(Product SomeProduct, int Quantity)

{

Console.WriteLine($"Прибуло {Quantity} {SomeProduct.Name}. Формування видаткової 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();

}

}

}

* 1. ***Консольний додаток:***
     1. ***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 післ 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();

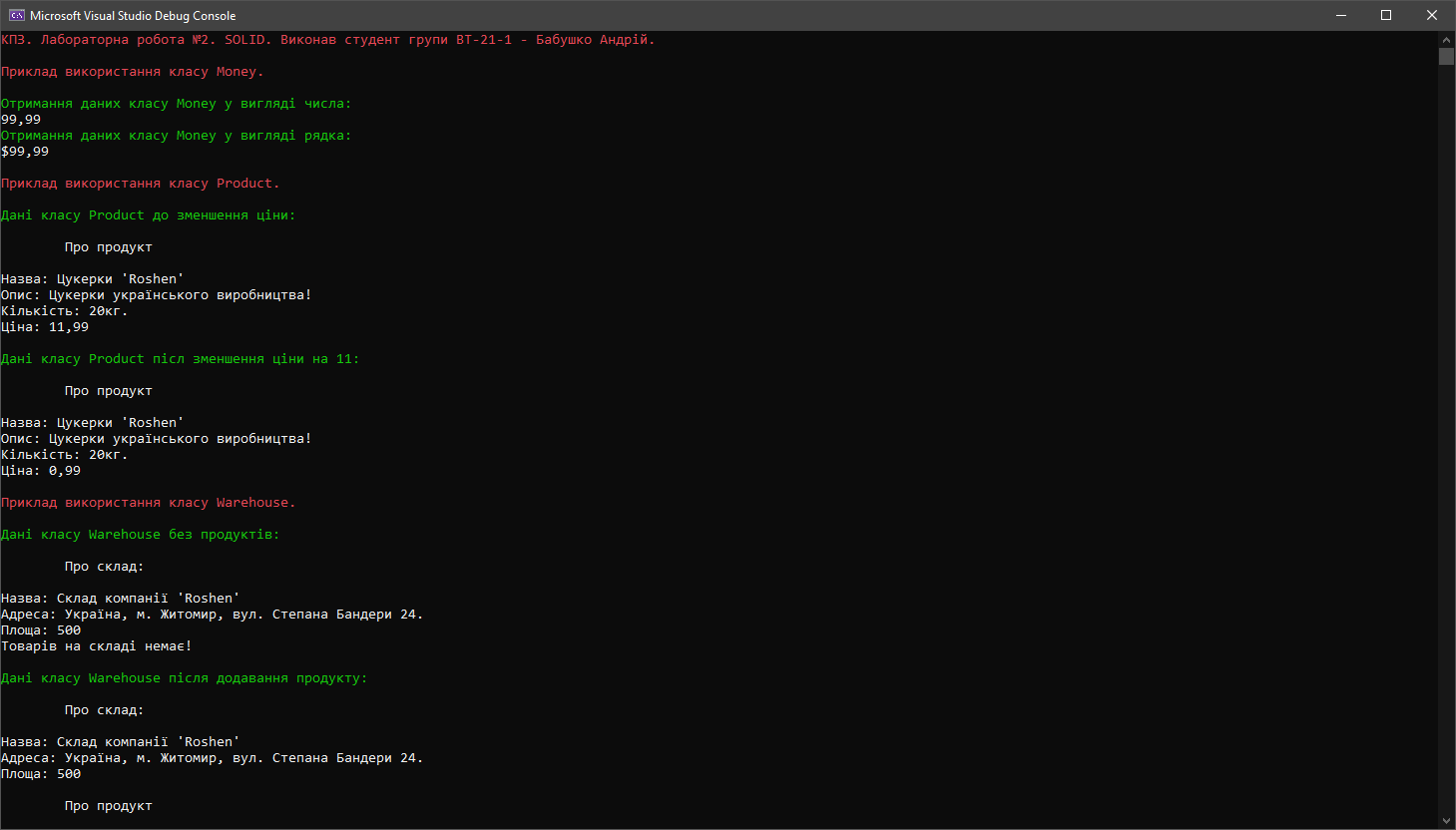
MyReporting.GenerateInventoryReport();

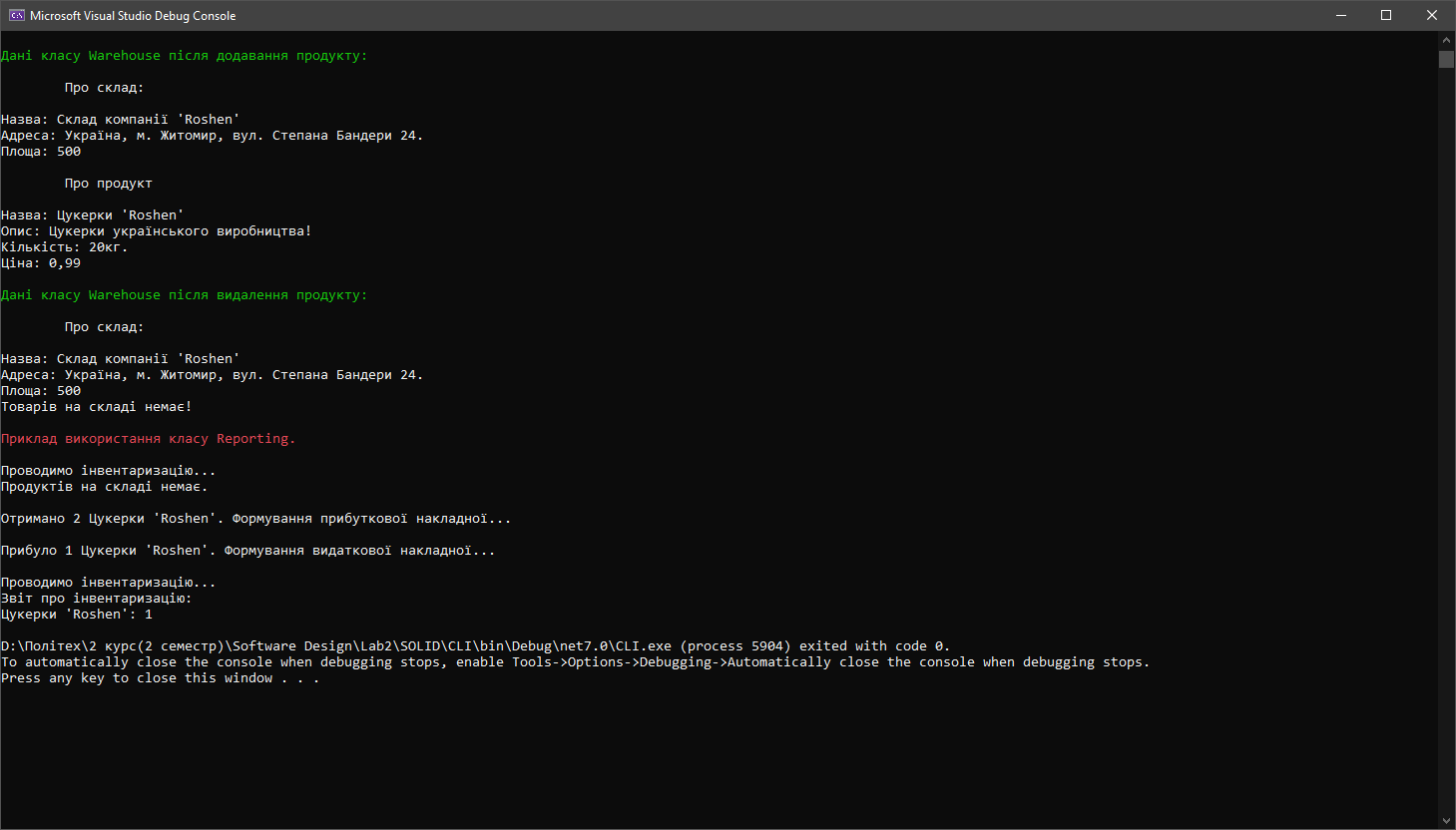
MyReporting.RegisterReceipt(MyProduct, 2);

MyReporting.RegisterShipment(MyProduct, 1);

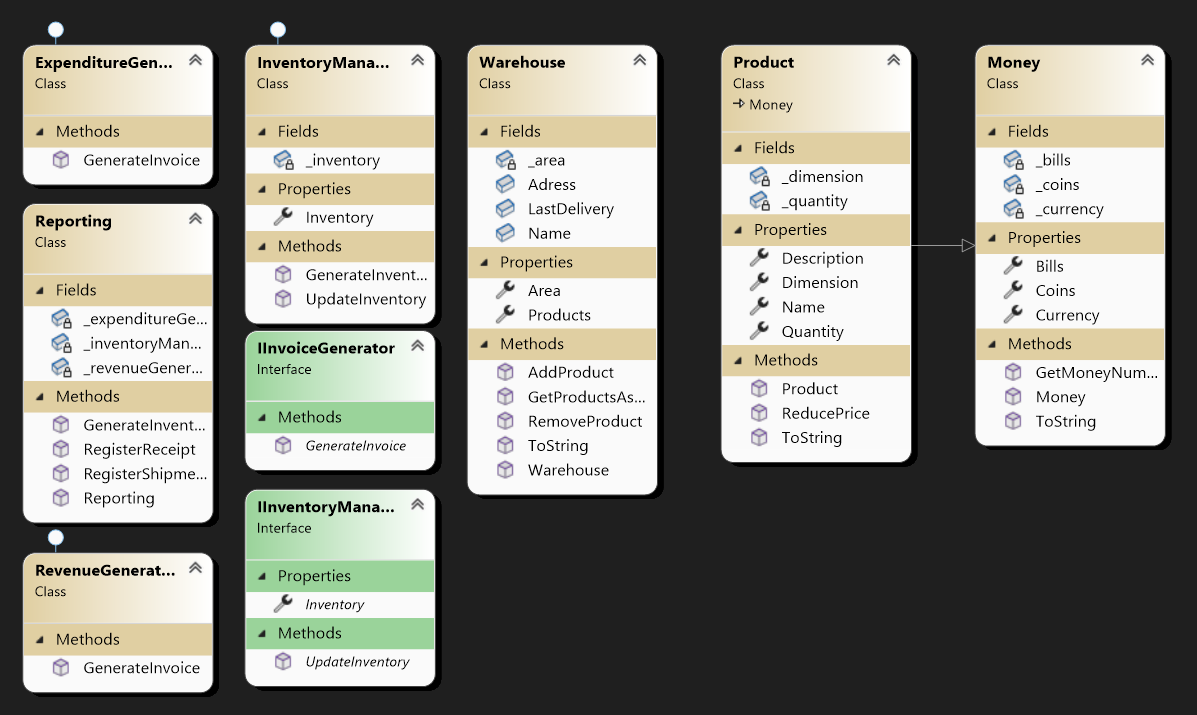
MyReporting.GenerateInventoryReport();

1. ***Результати програми:***

******

******

1. ***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.