**ЛАБОРАТОРНА РОБОТА № 2**

**Принципи програмування. SOLID**

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

**Хід роботи:**

1. **Демонстрація** роботи в Program.cs:

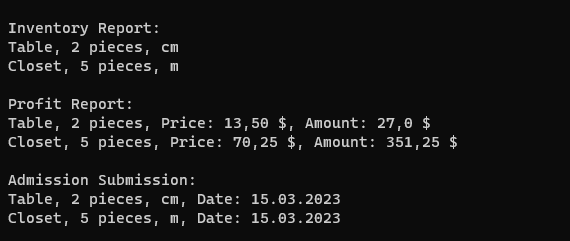


Рис. 1. Демонстрація звітів

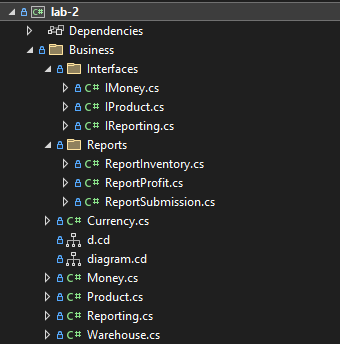


Рис. 2. Структура проекту

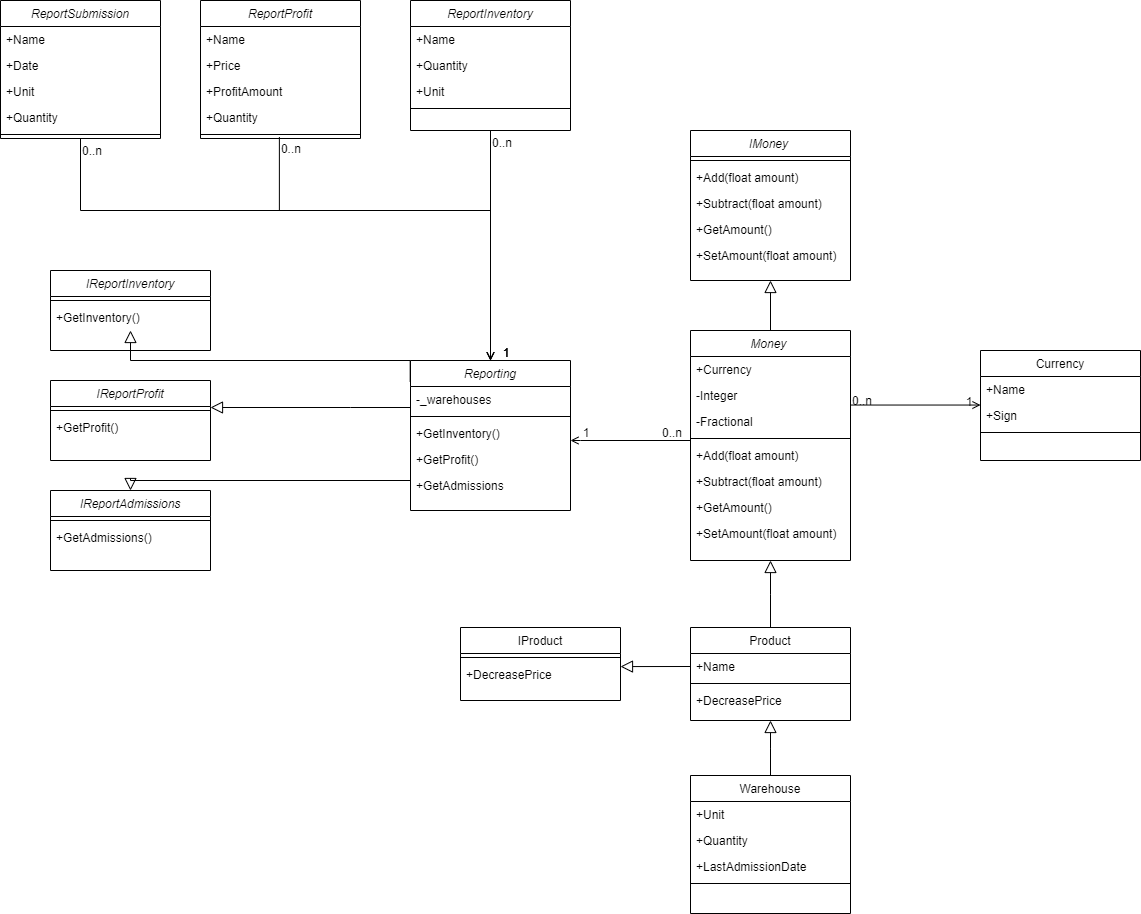


Рис. 3. UML діаграма

1. **Лістинг коду**

Program.cs:

using Business;

using Business.Structures;

Currency usd = new Currency("USD", '$');

Product product = new Product("Table", 13.50f, usd);

var tableWarehouse = new Warehouse("Table", "cm", 2, DateTime.Now, 13.50f, usd);

var closetWarehouse = new Warehouse("Closet", "m", 5, DateTime.Now, 70.25f, usd);

var warehouseList = new List<Warehouse>() { tableWarehouse, closetWarehouse };

Reporting reporting = new Reporting(warehouseList);

var inventoryReport = reporting.GetInventory();

var profitReport = reporting.GetProfits();

var submissionReport = reporting.GetSubmissions();

Console.WriteLine("\nInventory Report: ");

foreach (var item in inventoryReport)

{

Console.WriteLine($"{item.Name}, {item.Quantity} pieces, {item.Unit}");

}

Console.WriteLine("\nProfit Report: ");

foreach (var item in profitReport)

{

Console.WriteLine($"{item.Name}, {item.Quantity} pieces, Price: {item.Price}, Amount: {item.ProfitAmount}");

}

Console.WriteLine("\nAdmission Submission: ");

foreach (var item in submissionReport)

{

Console.WriteLine($"{item.Name}, {item.Quantity} pieces, {item.Unit}, Date: {item.Date.ToShortDateString()}");

}

Money.cs:

using Business.Interfaces;

using System.ComponentModel;

namespace Business

{

public class Money : IMoney

{

public Currency Currency { get; set; }

private uint Integer { get; set; }

private uint Fractional { get; set; }

public Money(Currency currency, float amount)

{

if (amount < 0)

{

throw new Exception("Money amount can't be a negative number");

}

SetAmount(amount);

Currency = currency;

}

public void Add(float amount)

{

var currentAmount = this.GetAmount();

currentAmount += amount;

SetAmount(currentAmount);

}

public void Subtract(float amount)

{

var currentAmount = this.GetAmount();

currentAmount -= amount;

SetAmount(currentAmount);

}

public float GetAmount()

{

var result = (float)Integer;

result += (float)(Fractional / 100.0);

return result;

}

public void SetAmount(float amount)

{

var parts = MoneyUtils.ParseFloat(amount);

Integer = parts.Item1;

Fractional = parts.Item2;

}

public override string ToString()

{

return $"{Integer},{Fractional} {Currency.Sign}";

}

}

public static class MoneyUtils

{

public static Tuple<uint, uint> ParseFloat(float amount)

{

var integer = (uint)amount;

var fractional = (uint)((amount % integer) \* 100);

return new Tuple<uint, uint>(integer, fractional);

}

}

}

Product.cs:

using Business.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Business

{

//Single Responsibility Principle.Each object has one responsibility and this responsibility is completely encapsulated in a class

//Liscov Substitution Principle. Objects of a superclass be replaceable with objects of its subclasses without breaking the application

//Open Closed Principle. Objects or entities open for extension but closed for modification

public class Product : Money, IProduct

{

public string Name { get; set; }

public Product(string name, float price, Currency currency) : base(currency, price)

{

Name = name;

}

public void DecreasePrice(float amount)

{

if (amount < 0)

{

throw new ArgumentException("Value can't be less than 0");

}

if (amount > GetAmount())

{

throw new ArgumentException("Price of the product can't be less than 0");

}

Subtract(amount);

}

}

}

Warehouse.cs:

using System;

using System.Collections.Generic;

using System.Formats.Asn1;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Business

{

//Single Responsibility Principle. Each object has one responsibility and this responsibility is completely encapsulated in a class

//Liscov Substitution Principle. Objects of a superclass be replaceable with objects of its subclasses without breaking the application

public class Warehouse : Product

{

public string Unit { get; set; }

public uint Quantity { get; set; }

public DateTime LastAdmissionDate { get; set; }

public Warehouse(string name, string unit, uint quantity, DateTime lastAdmissionDate, float price, Currency currency) : base(name, price, currency)

{

Unit = unit;

Quantity = quantity;

LastAdmissionDate = lastAdmissionDate;

}

}

}

Reporting.cs:

using Business.Interfaces;

using Business.Structures;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Business

{

//Single Responsibility Principle. Each object has one responsibility and this responsibility is completely encapsulated in a class

public class Reporting : IReportInventory, IReportProfit, IReportSubmissions

{

private List<Warehouse> \_warehouses { get; set; }

public Reporting(List<Warehouse> warehouses)

{

\_warehouses = warehouses;

}

public List<ReportInventory> GetInventory()

{

List<ReportInventory> reportList = new List<ReportInventory>();

foreach (var item in \_warehouses)

{

var reportInventory = new ReportInventory();

reportInventory.Name = item.Name;

reportInventory.Unit = item.Unit;

reportInventory.Quantity = item.Quantity;

reportList.Add(reportInventory);

}

return reportList;

}

public List<ReportProfit> GetProfits()

{

List<ReportProfit> reportList = new List<ReportProfit>();

foreach (var item in \_warehouses)

{

var reportProfit = new ReportProfit();

reportProfit.Name = item.Name;

reportProfit.Price = new Money(item.Currency, item.GetAmount());

reportProfit.Quantity = item.Quantity;

reportProfit.ProfitAmount = new Money(item.Currency, item.GetAmount()\*item.Quantity);

reportList.Add(reportProfit);

}

return reportList;

}

public List<ReportSubmission> GetSubmissions()

{

List<ReportSubmission> reportList = new List<ReportSubmission>();

foreach (var item in \_warehouses)

{

var reportSubmission = new ReportSubmission();

reportSubmission.Name = item.Name;

reportSubmission.Quantity = item.Quantity;

reportSubmission.Unit = item.Unit;

reportSubmission.Date = item.LastAdmissionDate;

reportList.Add(reportSubmission);

}

return reportList;

}

}

}

ReportInventory.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Business.Structures

{

public struct ReportInventory

{

public string Name { get; set; }

public uint Quantity { get; set; }

public string Unit { get; set; }

}

}

ReportProfit.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Business.Structures

{

public struct ReportProfit

{

public string Name { get; set; }

public Money Price { get; set; }

public uint Quantity { get; set; }

public Money ProfitAmount { get; set; }

}

}

ReportSubmission.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Business.Structures

{

public struct ReportSubmission

{

public string Name { get; set; }

public uint Quantity { get; set; }

public string Unit { get; set; }

public DateTime Date { get; set; }

}

}

IMoney.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Business.Interfaces

{

//Interface Segregation Principle, A client should never be forced to implement an interface that it doesn't use

public interface IMoney

{

public void Add(float amount);

public void Subtract(float amount);

public float GetAmount();

public void SetAmount(float amount);

}

}

IProducs.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Business.Interfaces

{

//Interface Segregation Principle, A client should never be forced to implement an interface that it doesn't use

public interface IProduct

{

public void DecreasePrice(float amount);

}

}

IReporting.cs:

using Business.Structures;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Business.Interfaces

{

//Interface Segregation Principle, A client should never be forced to implement an interface that it doesn't use

public interface IReportProfit

{

public List<ReportProfit> GetProfits();

}

public interface IReportSubmissions

{

public List<ReportSubmission> GetSubmissions();

}

public interface IReportInventory

{

public List <ReportInventory> GetInventory();

}

}

***Висновки:*** в ході лабораторної роботи навчився дотримуватися принципів SOLID та обґрунтовувати їх. Навчитися описувати дизайн програми за допомогою UML діаграм.