# ЛАБОРАТОРНА РОБОТА 1: КОМПОНЕНТНО-ОРІЄНТОВАНЕ ПРОЕКТУВАННЯ.

Мета роботи: Отримати практичні навички компонентно-орієнтованого проектування програмного забезпечення. Для цього, створити бібліотеку DLL.

## 1.1 Теоретичні відомості

Сучасні кросплатформні системи створюються з використанням *компонентно-орієнтованого підходу*.

*Компонентне програмування* є подальшим розвитком об'єктно-орієнтованого програмування (ООП) і концепції *«повторного використання»* (*reuse*).

Компонентне програмування покликане забезпечити простішу і швидшу процедуру інсталяції прикладного програмного забезпечення, а також збільшити відсоток повторного використання коду, тобто підсилити основні переваги ООП.

Перш за все, сформулюємо головне для даного підходу визначення компонента.

Під *компонентом* далі матимемо на увазі незалежний модуль програмного коду, призначений для повторного використання і *розгортання*.

Такий компонент є структурною одиницею програмної системи, що має чітко визначений *інтерфейс*.

Компонент може бути незалежно поставлений або не поставлений, доданий до складу деякої системи або видалений з неї, у тому числі, може

включатися до складу систем інших постачальників.

Таким чином, *компонент* — це виділена структурна одиниця розгортання з чітко визначеним інтерфейсом**.**

Всі залежності компонента від оточення мають бути описані в рамках цього інтерфейсу. Один компонент може також мати декілька інтерфейсів, граючи декілька різних ролей в системі. При описі інтерфейсу компонента важлива не лише сигнатура операцій, які можна виконувати з його допомогою. Стає важливим і те, які інші компоненти він може використовувати при роботі, а також яким обмеженням повинні задовольняти вхідні дані операцій і які властивості виконуються для результатів їх роботи. Ці обмеження є так званим *інтерфейсним контрактом* або *програмним контрактом компонента***.**

**Властивості компонентів**

***1. Використання компонента (виклик його методів) можливе лише через його інтерфейси відповідно до контракту***.

Інтерфейс компонента включає набір операцій, які можна викликати в будь-якого компонента, який реалізує даний інтерфейс, і набір операцій, які цей компонент може викликати у відповідь в інших компонентів. *Інтерфейсний контракт* для кожної операції самого компонента (або використовуваного ним) визначає правила взаємодії компонентів в системі.

Для опису інтерфейсів призначені мови опису інтерфейсів IDL (Interface Definition Language).

Ці мови містять *операції*, які є викликами відкритих (public) методів класів, що входять до складу компонента, і *операнди* – відкриті (public) поля класів.

***2. Компонент повинен працювати в будь-якому середовищі, де є необхідні для його роботи інші компоненти.***

Це вимагає наявності певної інфраструктури, яка дозволяє

компонентам знаходити один одного і взаємодіяти по певних правилах.

Набір правил визначення інтерфейсів компонентів і їх реалізацій, а також правил, за якими компоненти працюють в системі і взаємодіють один з одним, прийнято називати *компонентною моделлю* (component model).

У компонентну модель входять правила, що регламентують життєвий цикл компонента, тобто те, через які стани він проходить при своєму існуванні в рамках деякої системи (незавантажений, завантажений, пасивний, активний, знаходиться в кеші і ін.) і як виконуються переходи між цими станами).

Існують декілька компонентних моделей в різних ОС і від різних виробників. Правильно взаємодіяти один з одним можуть лише компоненти, побудовані в рамках *однієї моделі*, оскільки компонентна модель визначає «мову», на якій компоненти можуть спілкуватися один з одним.

Окрім компонентної моделі, для роботи компонентів необхідний деякий набір базових служб (basic services). Наприклад, компоненти повинні уміти знаходити один одного в середовищі, яке, можливо, розподілене на декілька машин. Компоненти повинні уміти передавати один одному дані, знову ж таки, можливо, за допомогою мережевих взаємодій, але реалізації окремих компонентів самі по собі не повинні залежати від вигляду використовуваного зв'язку і від розташування їх партнерів по взаємодії. Набір таких базових, необхідних для функціонування більшості компонентів служб, разом з підтримуваною з їх допомогою компонентною моделлю називається компонентним середовищем (або компонентним каркасом, component framework). Приклади відомих компонентних середовищ — різні реалізації J2EE, .NET Framework, CORBA, COM, COM+, DCOM.

***3. Компоненти відрізняються від класів об'єктно-орієнтованих мов.***

Клас визначає не лише набір інтерфейсів, що реалізуються, але і саму

їх реалізацію, оскільки він містить код визначуваних операцій. Контракт компонента, найчастіше, не фіксує реалізацію його інтерфейсів. Клас написаний на певній мові програмування. Компонент же не прив'язаний до певної мови, якщо, звичайно, його компонентна модель цього не вимагає, — компонентна модель є для компонентів тим же, чим для класів є мова програмування.

Зазвичай компонент є *крупнішою структурною одиницею*, ніж клас. Реалізація компонента часто складається з декількох тісно зв'язаних один з одним класів.

***4. Компоненти не залежать від мов програмування.***

Компоненти зберігаються і поширюються у бінарному вигляді і не залежать від мов програмування програмної системи.

Користувач і постачальник компонента можуть використовувати різні мови і бути територіально розподіленими.

## 1.2 Порядок виконання

### 1.2.1 Створення DLL-бібліотеки

Створення DLL-бібліотеки розглянемо на конкретному прикладі.

Запустіть Visual Studio 2008, із стартової сторінки перейдіть до створення проекту, виберіть тип проекту «**Class Library**». У вікні створення DLL всі поля заповнені значеннями за замовчанням. Як правило, їх слід перевизначити, задаючи власну інформацію.

У полі Name задайте ім'я DLL –бібліотеки, наприклад, MyLib.

У полі Location вкажіть шлях до каталогу, де зберігатиметься Рішення, що містить проект.

Розмістіть рішення в папці Lab1.

У полі Solution вибраний елемент «Create New Solution», що створює нове Рішення. Альтернативою є елемент списку, який вказує,

що проект може бути доданий до існуючого Рішення.

У вікні Solution Name задано ім'я Рішення. Виберіть Lab1.

Зверніть увагу на інші налаштування, зроблені в цьому вікні, - включений прапорець (за замовчанням) «Create directory for solution», у верхньому віконці із списку можливих каркасів вибраний каркас Framework .Net 3.5. Задавши необхідні установки і клацнувши по кнопці «OK», отримаємо автоматично побудований проект DLL, відкритий в середовищі Visual Studio 2008 .

Імена класів повинні бути змістовними. Змініть ім'я «Class1» на ім'я «MySin». Для цього у вікні коду проекту виділіть ім'я змінної об'єкту, потім в головному меню виберіть пункт *Refactor* і підпункт *Rename*. У вікні, що відкрилося, вкажіть нове ім'я. Тоді будуть показані всі місця, що вимагають перейменування об'єкту. В нашому прикладі буде лише одна очевидна заміна, але в загальному випадку замін багато, так що автоматична заміна всіх входжень корисна.

Наступний крок також продиктований правилом стилю – ім'я класу і ім'я файлу, що зберігає клас, повинні бути однаковими. Перейменування імені файлу робиться безпосередньо у вікні проектів Solution Explorer.

І наступний крок, продиктований також важливим правилом стилю, - додавання коментаря. Для цього в рядку перед заголовком класу слід набрати три слеша (три косі риски). В результаті перед заголовком класу з'явиться заголовний коментар – тег «summary», в який і слід додати короткий, але змістовний опис призначення класу. Теги «summary», якими слід супроводжувати класи, відкриті (public) методи і поля класу відіграють три важливі ролі. Вони полегшують розробку і супровід проекту, роблячи його самодокументованим. Клієнти класу при створенні об'єктів класу отримують інтелектуальну підказку, що

пояснює суть того, що можна робити з об'єктами. Спеціальний інструментарій дозволяє побудувати документацію за проектом, що включає інформацію з тегів «summary».

У нашому випадку коментар до класу MySin може бути достатньо простим – «Обчислення математичних функцій».

Напишемо реалізацію одного методу класу – обчислення функції Sin(x) через ряд Тейлора.

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Спочатку напишемо коментарі до методу (у форматі XML). Далі напишемо реалізацію методу. Отримаємо код.

Лістинг 1.1 – Код бібліотечного методу

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace MyLib
{
///Обчислення математичних функцій

    public class MySin
    {
     /// <summary>
     /// Sin(x)
     /// </summary>
```

```
        /// <param name="x">кут в радіанах – перший
аргумент функції  Sin</param>
        ///<param name="n">показник ступеня – другий
аргумент функції  Sin</param>
        /// <returns>Повертає значення функції Sin для
заданого кута</returns>
        public static double Sin(double x, int n)
        {
            double result = 0;
            for (int i = 0; i < n; i++)
            {
                result = result + (Math.Pow((-1), i) *
Math.Pow(x, (2 * i + 1))) / F(2 * i + 1);
            }
            return result;
        }
        static double F(int n)
        {
            double tmp = 1;
            for (int i = 1; i <= n; i++)
            {
                tmp = tmp * i;
            }
            return tmp;
        }
    }
}
```

Побудуємо Рішення, що містить проект, для чого в Головному меню виберемо пункт Build|Build Solution. В результаті успішної

компіляції буде побудований файл з розширенням dll. Оскільки побудована збірка не містить виконуваного файлу, то безпосередньо запустити наш проект на виконання не вдасться. Побудуємо консольний проект, до якого приєднаємо нашу DLL, і протестуємо, наскільки коректно працюють створені нами методи.

### 1.2.2 Створення консольного проекту для тестування функції з бібліотеки

Виберіть пункт меню File|New|Project, задайте тип проекту ConsoleApplication, назвіть йому – ConsoleMySin, вкажіть, що проект додається до існуючого Рішення Lab1. В результаті у вже існуюче Рішення додасться ще один проект.

Напишіть код, який викликає реалізовану функцію Sin(x,n), стандартну функцію Sin(x), обчислює похибку і виводить результат на консоль.

Лістинг 2.2 – Консольний застосунок, який викликає бібліотечний метод

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleMySin
{
    class Program
    {
        /// <summary>
```

```
        /// Виклик бібліотечного методу Sin(x,n) з
MySin.dll
        /// </summary>
        /// <param name="args"></param>
        static void Main(string[] args)
        {
            Console.WriteLine("Введите x- угол в
радианах");
            double x =
double.Parse(Console.ReadLine());
            Console.WriteLine("Введите показатель
степени n");
            int n = int.Parse(Console.ReadLine());
            //вызов метода вычисления sin(x) из
библиотеки
            double my_sinus = MyLib.MySin.Sin(x, n);
            //вызов метода из класса Math
            double sinus = Math.Sin(x);
            double delta = sinus - my_sinus;
            Console.WriteLine("my_sinus=
{0},sin={1},delta={2}", my_sinus, sinus, delta);
            Console.ReadKey();
        }
    }
}
```

Побудуємо рішення і отримаємо *повідомлення про помилку.* Наша бібліотека не підключена до проекту.

***Підключення проекту бібліотеки до консольного проекту.***

Для цього додайте посилання на проект з DLL MySin. У вікні

Solution Explorer наведіть покажчик миші до імені консольного проекту і з контекстного меню виберіть пункт меню «Add Reference». Виберіть вкладку «Projects». Оскільки проект MySin включений в Рішення, то він автоматично з'явиться у вікні, Якщо посилання потрібно встановити на проект, не включений в Рішення, то у вікні додавання посилань потрібно вказати шлях до проекту.

Посилання на DLL з'явиться в папці «References» консольного проекту. Тепер проекти зв'язані і з консольного проекту доступний реалізований бібліотечний метод.

Перебудуйте рішення, щоб не було помилок.

***Встановлення стартового проекту.***

У вікні Solution Explorer наведіть курсор миші на заголовок консольного проекту і виберіть: *Set as StartUp Project*

Після цього його можна запустити на виконання.

### 1.2.3 Створення Windows-проекту в тому самому рішенні

Виберіть пункт меню File|New|Project, задайте тип *проекту Windows Forms Application***,** назвіть його – WindowsMySin, вкажіть, що проект додається до існуючого Рішення *Lab1*.

На формі створіть 2 текстові поля для введення вхідних параметрів, третє і четверте – для результатів (рисунок 2.1).

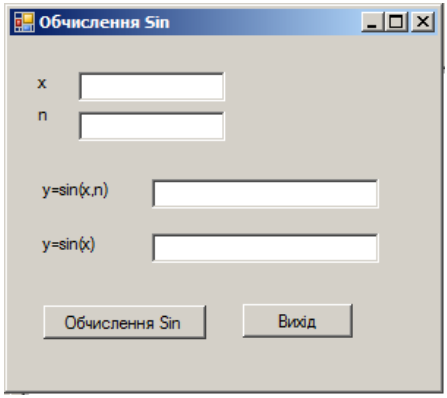


Рис. 2.1 – Форма для обчислення sin(x)

Додамо 2 кнопки. При натисканні кнопки ***"Обчислення Sin"*** виконується виклик функцій, *"Вихід"* – завершення роботи.

Лістинг 2.3 – Код форми

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsMySin
{
    public partial class Form1 : Form
    {
        public Form1()
```

```
        {
            InitializeComponent();
        }

        private void button1_Click(object sender,
EventArgs e)
        {
            double x = double.Parse(txt_x.Text);
            int n = int.Parse(txt_n.Text);
            //виклик метода обчислення sin(x) із
бібліотеки
            double my_sinus = MyLib.MySin.Sin(x, n);
            //вызов метода из класса Math
            double sinus = Math.Sin(x);
            txt_y1.Text = my_sinus.ToString();
            txt_y2.Text = sinus.ToString();

        }

        private void button2_Click(object sender,
EventArgs e)
        {
            this.Close();
        }
    }
}
```

Робимо проект стартовим і запускаємо на виконання. Результат:

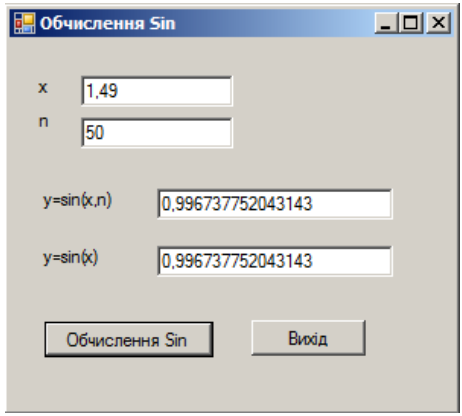


Рис. 2.2 – Результат роботи форми

## 1.3 Порядок захисту

Студент, який виконав завдання лабораторної роботи, подає письмовий звіт про виконання лабораторної роботи і доповідає про результати та дає відповіді на запитання. Виконане завдання оцінюється згідно системи контролю знань, умінь, навичок, наведеної на відповідній сторінці дистанційного курсу.

## 1.4 Контрольні запитання

1. Що таке компонентна модель і яке її призначення?
2. Що таке DLL-бібліотека? Чим вона відрізняється від консольного проекту?
3. Чому бібліотечні методи повинні визначатися з модифікатором доступу public?
4. Що є інтерфейсним контрактом бібліотеки?

## 1.5 Перелік джерел


1. Парфьонов Ю. Е., Поляков А. О. Робоча програма навчальної дисципліни "КРОСПЛАТФОРМОВІ ТА БАГАТОЛАНКОВІ ТЕХНОЛОГІЇ" Харків. Вид. ХНЕУ, 2012
2. Макарова Н.В., Волков В.Б. Информатика: Учебник для вузов. – СПб.: Питер, 2011. – 576 с.: ил.
3. Іванніков Є.Ю. Послуга повної довірчої конфіденційності для захищеної ОС на базі GNU/LINUX з рошириенням RSBAC//Проблеми програмування.-2010.-№3 .-с.513-518
4. Анісімов А.В., Іванніков Є.Ю. Послуга "КО-1. Повторне використання об’єктів" для захищеної ОС на базі GNU/LINUX з розширенням RSBAC//Проблеми програмування.-2010.-№4 .-с.11-20
5. Блинов И. Н. Java. Промышленное программирование : практ. пособ. / И. Н. Блинов, В. С. Романчик. – Мн : УниверсалПресс, 2007. – 768 с.
6. Дейтел Х. М. Технологии программирования на Java 2: Книга 2. Распределенные приложения / Х. М. Дейтел, П. Дж. Дейтел. ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 464 с. : ил.
7. Дейтел Х. М. Технологии программирования на Java 2: Книга 3. Корпоративные системы / Х. М. Дейтел, П. Дж. Дейтел, С. И. Самтри ; пер. с англ. – М. : ООО "Бином-Пресс", 2003. – 672 с. : ил.
8. Дэвид М. Гери, JavaServer Faces. Библиотека профессионала. JavaServer Faces. CORE / Дэвид М. Гери, Кей С. Хорстманн. – 3-е изд. – М. : Издательский дом "Вильямс", 2011. – 544 с.
   Брюс Еккель,Thinking in Java.,пер.Є.Матвєєв.Бібліотека програміста, в-во "Пітер", 2009 - 640 с.