КИЇВСЬКИЙ ФАХОВИЙ КОЛЕДЖ ЗВ’ЯЗКУ

Циклова комісія «Програмної інженерії та прикладної математики»

З**ВІT**

**ПРО ВИКОНАННЯ НАВЧАЛЬНОЇ**

**ПРАКТИКИ З ОБ’ЄКТНО-ОРІЄНТОВАНОГО**

ПРОГРАМУВАННЯ

**Студента** Дешпетка Максима Олександровича\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

**Групи** \_\_\_\_РПЗ-93б\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Спеціальності** \_ 121 – «Інженерія програмного забезпечення»\_\_\_\_\_\_

**Кваліфікаційний рівень** молодший бакалавр\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**База практики** \_\_Київський фаховий коледж зв’язку\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Керівник практики** \_\_Ольшевська З.О.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище, ім’я, по батькові)

Звіт захищено «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 р.

Оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Київ 2022

**ЗМІСТ**

[ВСТУП 3](#Вступ)

[1 ПРАКТИЧНА РОБОТА №1 5](#пра1)

[Завдання 1: 5](#пра11)

[Завдання 2: 5](#пра12)

[Завдання 3: 5](#пра13)

[Завдання 4: 5](#пра14)

[Завдання 5: 5](#пра15)

[Завдання 6: 5](#пра16)

[2 ПРАКТИЧНА РОБОТА №2 14](#пра2)

[Завдання 1: 14](#пра21)

[3 ПРАКТИЧНА РОБОТА №3 17](#пра3)

[Завдання 1: 17](#пра31)

[4 ПРАКТИЧНА РОБОТА №4 22](#пра4)

[Завдання 1: 22](#пра41)

[5 ПРАКТИЧНА РОБОТА №5 25](#пра5)

[Завдання 1: 25](#пра51)

[Завдання 2: 25](#пра52)

[6 ПРАКТИЧНА РОБОТА №6 36](#пра6)

[Завдання 1: 36](#пра61)

[7 ПРАКТИЧНА РОБОТА №7 39](#пра7)

[Завдання 1: 39](#пра71)

[Завдання 2: 39](#пра72)

[Завдання 3: 39](#пра73)

[8 ПРАКТИЧНА РОБОТА №8 43](#пра8)

[Завдання 1: 43](#пра81)

[Завдання 2: 43](#пра82)

[9 ПРАКТИЧНА РОБОТА №9 52](#пра9)

[Завдання 1: 52](#пра91)

[ВИСНОВКИ 58](#_Toc104385851)

[ПЕРЕЛІК ПОСИЛАНЬ 59](#_Toc104385852)

[ЩОДЕННИК ПРАКТИКАНТА 60](#_Toc104385853)

[Критерії обліку знань студентів 61](#_Toc104385854)

# ВСТУП

Дана навчальна практика виконана на мові програмування C#.

C# — об’єктно-орієнтована мова програмування. Розроблена в 1998—2001 роках групою інженерів під керівництвом Андерса Хейлсберга в компанії Microsoft як мова розробки додатків для платформи Microsoft .NET Framework та потім була стандартизована як ECMA-334 и ISO/IEC 23270.

С# взяв краще від мови програмування C (його високу продуктивність), C++ (об'єктно-орієнтовану структуру), Java (високу безпеку та збирач сміття). На C# можна створити динамічну сторінку, службу XML, компонент доступу до баз даних, десктопну програму під Windows та клієнтську програму або її компонент. Мова програмування C# підходить не тільки для написання мережевих та веб-додатків, вона надає засоби для створення будь-якого типу компонентів для платформи Windows.

До можливостей C# також належать: повна підтримка класів та ООП; узгоджений та чітко визначений набір базових типів; підтримка автоматичного створення XML-документації; підтримка властивостей та подій у стилі Visual Basic; можливість використання для написання динамічних веб-сторінок ASP.NET та веб-служб XML.

Навчальна практика виконується протягом 4 тижнів і охоплює основні теми предмету «Об’єктно-орієнтованого програмування». Теоретичною основою для виконання навчальної практики є курс лекцій, лабораторні роботи, практичні заняття та навчальна література.

**Практична робота № 1. «Розробка класу «Раціональне число»**

**Мета роботи:** засвоїти основи створення класів в мові C#, набути

практичних навичок застосування принципів поліморфізму об’єктно-

орієнтованого програмування.

**Завдання**

1. Доповнити клас ***Fraction*** перевантаженими арифметичними операціями, у яких один із операндів є дійсним числом.
2. Доповнити клас ***Fraction*** перевантаженими операціями порівняння для дробив та дійсних чисел.
3. Доповнити клас ***Fraction*** перевантаженим конструктором, який здійснює перетворення дійсного числа до типу ***Fraction***. Передбачається, що дробова частина речового числа містить до 10 знаків після коми.
4. Розробити клас "Комплексне число". Визначити у ньому конструктор, перевантажити арифметичні операції, операції порівняння, операцію перетворення на рядок та статичний метод отримання комплексного числа з рядка.
5. Розробити клас "Дата". Визначити у ньому конструктори та деструктор, перевантажити операцію додавання до дати заданої кількості днів, операцію віднімання двох дат, операції порівняння та операцію перетворення на символьний рядок, а також статичний метод отримання дати з рядка.
6. Розробити клас "Час". Визначити в ньому конструктори та деструктор, перевантажити операцію додавання до часу заданої кількості хвилин, операцію віднімання двох моментів часу, операцію перетворення на символьний рядок та метод отримання моменту часу з рядка.

1.

У класі **Fraction** визначено конструктор, який ініціалізує поля з чисельником, знаменником і цілою частиною. Також у ньому виконується метод **Cut:**

public float fraction;

public long numerator;

public long denominator;

public Fraction(float fraction, long numerator, long denominator)

{

this.fraction = fraction;

this.numerator = numerator;

this.denominator = denominator;

Cut();

}

public Fraction(Fraction f) : this(f.fraction, f.numerator, f.denominator)

{ }

public Fraction(double a) : this((Fraction)a)

{ }

Метод який приводить дріб до простої форми.

public void NormalView()

{

numerator += denominator \* fraction;

fraction = 0;

if (numerator < 0 && denominator < 0)

{

numerator \*= -1;

denominator \*= -1;

}

Cut();

}

Метод який приводить дріб до неправильної форми.

public Fraction MixedView()

{

fraction = (long)Math.Floor(Convert.ToDouble(numerator) / Convert.ToDouble(denominator));

numerator -= fraction \* denominator;

return this;

}

Метод Cut реалізує знаходження спільного дільника і скорочує дріб.

public void Cut()

{

long a = this.numerator;

long b = this.denominator;

while (a != 0 && b!= 0)

{

if (a > b)

a = a % b;

else

b = b % a;

}

long cut = a + b;

numerator /= cut;

denominator /= cut;

}

3.

Далі статичний метод перевантаження операції явного перетворення з типу float до типу Fraction.

public static explicit operator Fraction(float number)

{

if (number % 1 == 0)

return new Fraction(0, (long)number, 1);

float decimals = number.ToString().Split(',')[1].Length;

long den = (long)Math.Pow(10, decimals%11);

long num = (long)(number \* den);

return new Fraction(0, num, den);

}

Наступна перевантажена операції неявного перетворення з типу Fraction до типу float.

public static implicit operator float(Fraction number)

{

return ((float)(number.numerator) + number.fraction \* number.denominator) / number.denominator;

}

Метод арифметичної операції додавання дробів.

public static Fraction operator +(Fraction f1, Fraction f2)

{

f1.NormalView();

f2.NormalView();

var f3 = new Fraction(0, 1, 1);

f3.numerator = f1.numerator \* f2.denominator + f2.numerator \* f1.denominator;

f3.denominator = f1.denominator \* f2.denominator;

f3.NormalView();

return f3;

}

Метод арифметичної операції віднімання дробів.

public static Fraction operator -(Fraction f1, Fraction f2)

{

return f1 + new Fraction(f2.fraction, -f2.numerator, f2.denominator);

}

Метод арифметичної операції множення дробів.

public static Fraction operator \*(Fraction f1, Fraction f2)

{

var f3 = new Fraction(0, 1, 1);

f1.NormalView();

f2.NormalView();

f3.numerator = f1.numerator \* f2.numerator;

f3.denominator = f1.denominator \* f2.denominator;

f3.NormalView();

return f3;

}

Метод арифметичної операції ділення дробів.

public static Fraction operator /(Fraction f1, Fraction f2)

{

long num\_b = f2.numerator;

f2.numerator = f2.denominator;

f2.denominator = num\_b;

return f1 \* f2;

}

Метод претворення дробу до рядка.

public override string ToString()

{

return $"{(fraction != 0 ? fraction + " " : "")}{numerator}/{denominator}";

}

1. Розробити клас "Комплексне число". Визначити у ньому конструктор, перевантажити арифметичні операції, операції порівняння, операцію перетворення на рядок та статичний метод отримання комплексного числа з рядка.

Лістинінг коду класу "Комплексне число":

class ComplexNumber

{

public float real;

public float imaginary;

public ComplexNumber(float real, float imaginary)

{

this.real = real;

this.imaginary = imaginary;

}

Конструктор комплексного числа

static public ComplexNumber operator +(ComplexNumber n1, ComplexNumber n2)

{

return new ComplexNumber(n1.real + n2.real, n1.imaginary + n2.imaginary);

}

**Додовання комплексних чисел.** Сумою двох комплексних чисел z1=a+bi та z2=c+di є комплексне число z1+z2 = a+c+i(b+d). Таким чином, реальні та уявні частини комплексних чисел складаються при складанні. Слід зазначити, що з перестановки місць доданків z1+z2 сума не зміниться.

static public ComplexNumber operator -(ComplexNumber n1, ComplexNumber n2)

{

n2.real \*= -1;

n2.imaginary \*= -1;

return n1 + n2;

}

**Віднімання комплексних чисел** Додавання та віднімання відбуваються за правилом (a + bi) ± (c + di) = (a ± c) + (b ± d)i. Тому просто помножуємо на -1, а потім додаємо.

static public ComplexNumber operator \*(ComplexNumber n1, ComplexNumber n2)

{

var n3 = new ComplexNumber(0, 0);

n3.real = n1.real \* n2.real - n1.imaginary\*n2.imaginary;

n3.imaginary = n1.real \* n2.imaginary + n2.real \* n1.imaginary;

return n3;

}

**Добуток комплексних чисел**. Визначення добутку комплексних чисел встановлюється з таким розрахунком, щоб числа a + b·i та a′ + b′·i можна було перемножувати як алгебраїчні двочлени, і щоб число i мало властивість i2=−1 Добуток комплексних чисел a + b·i, і a′ + b′·i дорівнює:

(a +ib)⋅(a′ +ib′)=(a⋅a′ −b⋅b′) +i(a⋅b′ +b⋅a′).

**Ділення комплексних чисел.**

static public ComplexNumber operator /(ComplexNumber n1, ComplexNumber n2)

{

float denominator = (n2 \* new ComplexNumber(n2.real, -n2.imaginary)).real;

n2.imaginary \*= -1;

var n3 = n1 \* n2;

n3.real = (float)Math.Round(n3.real/denominator, 1);

n3.imaginary = (float)Math.Round(n3.imaginary / denominator, 1);

return n3;

}

Операція перетворення комплексного числа на рядок.

public string toString()

{

string symb = imaginary >= 0 ? "+" : "";

return $"{this.real} {symb}{imaginary}i";

}

Статичний метод отримання комплексного числа з рядка:

static public ComplexNumber toComplex(string str)

{

var n = new ComplexNumber(0, 0);

n.real = Int32.Parse(str.Split(' ')[0]);

n.imaginary= Int32.Parse(str.Split(' ')[1].Trim('i'));

return n;

}

}

1. Розробити клас "Дата". Визначити у ньому конструктори та деструктор, перевантажити операцію додавання до дати заданої кількості днів, операцію віднімання двох дат, операції порівняння та операцію перетворення на символьний рядок, а також статичний метод отримання дати з рядка.

Лістинінг коду класу "Дата":

class myDate

{

public int year;

public int month;

public int day;

public myDate(int year, int month, int day)

{

this.year = year;

this.month = month;

this.day = day;

}

~myDate()

{

Console.WriteLine("myDate Deleted");

}

Додавання двух дат.

static public myDate operator +(myDate d1, myDate d2)

{

int add\_m = (int)Math.Floor((d1.day + d2.day) / 31.0);

int add\_y = (int)Math.Floor((d1.month + d2.month + add\_m) / 12.0);

int y = add\_y + d1.year + d2.year;

return new myDate(y, (d1.month + d2.month + add\_m) % 12, (d1.day + d2.day) % 31);

}

Віднімання двух дат.

static public myDate operator -(myDate d1, myDate d2)

{

int d = d1.day - d2.day;

int min\_m = (int)Math.Floor(d / 31.0);

int m = d1.month - d2.month + min\_m;

int min\_y = (int)Math.Floor(m / 12.0);

int y = d1.year - d2.year + min\_y;

d = d <= 0 ? 31 + d : d;

m = m <= 0 ? 12 + m : m;

return new myDate(Math.Max(0, y), m, d);

}

public string toString()

{

return $"{year}/{month}/{day}";

}

static public myDate toDate(string str)

{

List<int> v = Array.ConvertAll(str.Split('/').ToArray(), int.Parse).ToList();

return new myDate(v[0], v[1], v[2]);

}

static public implicit operator int(myDate d)

{

return (d.year\*12+d.month)\*31+d.day;

}

}

1. Розробити клас "Час". Визначити в ньому конструктори та деструктор, перевантажити операцію додавання до часу заданої кількості хвилин, операцію віднімання двох моментів часу, операцію перетворення на символьний рядок та метод отримання моменту часу з рядка.

Лістинінг до класу “Час”:

class myTime

{

public int h, m, s;

public myTime(int h, int m, int s)

{

this.h = h;

this.m = m;

this.s = s;

}

static public myTime operator +(myTime t1, myTime t2)

{

return new myTime(t1.h + t2.h, t1.m + t2.m, t1.s + t2.s);

}

static public myTime operator -(myTime t1, myTime t2)

{

int s = t1.s - t2.s;

int min\_m = (int)Math.Floor(s / 60.0);

int m = t1.m - t2.m + min\_m;

int min\_h = (int)Math.Floor(m / 60.0);

int h = Math.Max(0, t1.h - t2.h + min\_h);

s = s <= 0 ? 60 \* -min\_m + s : s;

m = m <= 0 ? 60 \* -min\_h + m : m;

return new myTime(h, m, s);

}

public string toString()

{

return $"{(h > 9 ? h : "0" + h)}:{(m > 9 ? m : "0" + m)}:{(s > 9 ? s : "0" + s)}";

}

static public myTime toTime(string str)

{

List<int> v = Array.ConvertAll(str.Split(':').ToArray(), int.Parse).ToList();

return new myTime(v[0], v[1], v[2]);

}

~myTime()

{

Console.WriteLine("myTime deleted");

}

}

Лістинінг до класу Program:

class Program

{

static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.Unicode;

Console.InputEncoding = System.Text.Encoding.Unicode;

Console.WriteLine("Демонстрація класу дробу");

Fraction f1 = new Fraction(0, 100, 45);

Fraction f2 = new Fraction(1.2);

Fraction f3 = f1 + f2;

Console.WriteLine("1 дріб :");

f1.Print();

Console.WriteLine("2 дріб :");

f2.Print();

Console.WriteLine($"{f1} + {f2} = {f3}");

Console.WriteLine($"Виділення цілої частини : {f1 + f2} = {f3.MixedView()}");

Console.WriteLine("Чи більша 1 дріб ніж 2 :");

Console.WriteLine(f1 > f2);

Console.WriteLine("\nДемонстрація класу комплексного числа");

ComplexNumber c1 = new ComplexNumber(2, 3);

ComplexNumber c2 = new ComplexNumber(4, -5);

ComplexNumber c3 = ComplexNumber.toComplex("2 +4i");

Console.WriteLine("1 комплексне число :");

Console.WriteLine(c1);

Console.WriteLine("2 комплексне число :");

Console.WriteLine(c2);

Console.WriteLine($"{c1} / {c2} =");

Console.WriteLine((c1 / c2));

Console.WriteLine("\nДемонстрація класу дати");

myDate d1 = new myDate(1995, 9, 12);

myDate d2 = new myDate(0, 0, 31);

myDate d3 = myDate.toDate("2022/12/11");

Console.WriteLine($"d1 = {d1}\nd2 = {d2}");

Console.WriteLine($"{d1} - {d2} = {(d1 - d2)}");

Console.WriteLine($"d3 = {d3}");

Console.WriteLine($"{d1} < {d2} = {d1 < d2}");

Console.WriteLine("\nДемонстрація класу часу");

myTime t1 = myTime.toTime("1:1:35");

myTime t2 = myTime.toTime("0:0:150");

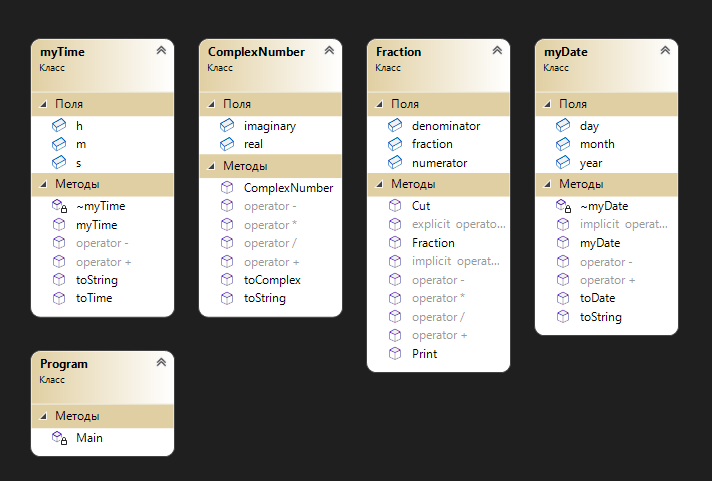
Console.WriteLine( $"{t1} - {t2} = {(t1 - t2)}");

Console.ReadLine();

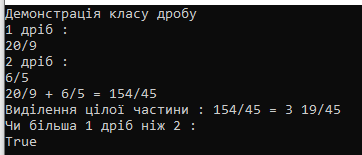
}

}

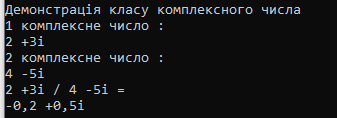
**UML-діаграма класів проекту**

****

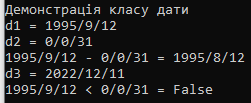
**Демонстрація роботи класів**



Робота класу *Fraction*

****

Робота класу *ComplexNumber*



Робота класу *myDate*



Робота класу *myTime*

**Практична робота № 2. «Використання колекцій. Класи ArrayList,Queue, Stack, Hashtable, SortedList»**

**Мета роботи**: засвоїти основи використання масивів і колекцій в мові C#, набути практичних навичок роботи з двовимірними масивами.

***Варіант 9.*** Для заданої матриці розміром NхN знайти таке k, що k-й рядок матриці співпадає з k-м стовпцем.

Лістинінг програми:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace weee

{

class Program

{

static void Main(string[] args)

{

var rand = new Random();

List<List<int>> mat = new List<List<int>>();

int copy\_c = rand.Next() % 4;

int copy\_r = rand.Next() % 4;

int N = Math.Max(copy\_r, copy\_c) + rand.Next() % 5+1;

Створення двовимірного списку та його заповнення.

Console.WriteLine($"Copy column: {copy\_c}, Copy row: {copy\_r}");

List<int> copy\_values = new List<int>();

for (int i = 0; i < N; i++)

copy\_values.Add(rand.Next()%10);

copy\_values[copy\_c] = copy\_values[copy\_r];

Спеціальна генерація однакового рядка та стовбця.

Console.WriteLine($"Initialize matrix...");

for (int i=0;i<N;i++)

{

mat.Add(new List<int>());

for (int j = 0; j < N; j++)

{

mat[i].Add(rand.Next() % 10);

bool copy = false;

if (i == copy\_r)

{

mat[i][j] = copy\_values[j];

copy = true;

}

if (j == copy\_c)

{

mat[i][j] = copy\_values[i];

copy = true;

}

Console.ForegroundColor = copy ? ConsoleColor.Green : ConsoleColor.White;

Console.Write(mat[i][j] + " ");

}

Console.WriteLine();

}

Заповнення двовимірного списку рандомними значеннями з урахуванням того, що в нас вже є заготовлені однакові стовбець та рядок.

Console.WriteLine($"Done!\n");

Console.WriteLine($"Start search for k =>");

for (int row = 0; row < N; row++)

{

for (int col = 0; col < N; col++)

{

bool same = true;

for (int i = 0; i < N; i++)

{

if (mat[row][i] != mat[i][col])

{

same = false;

break;

}

}

if (same)

{

Console.ForegroundColor = ConsoleColor.Blue;

Console.WriteLine($"Find same -> row: {row}, column: {col}");

Console.ForegroundColor = ConsoleColor.White;

}

else

Console.WriteLine($"Don't same -> row: {row}, column: {col}");

}

}

Пошук однакових стовпця та рядка серед усіх інших в списку. При успіху програма повідомить про однакові стовпець та рядок.

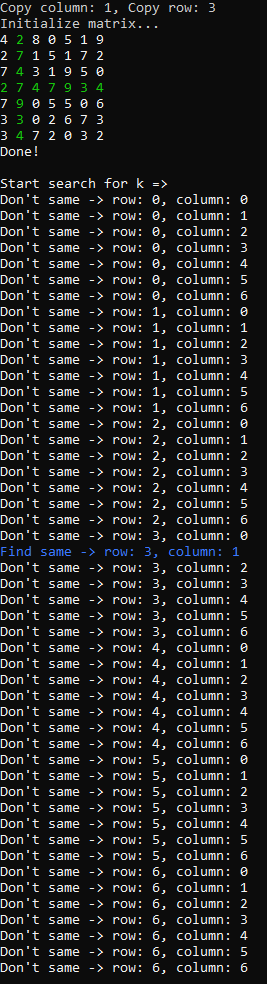
Console.ReadLine();

}

}

}

**Демонстрація роботи програми**



**Практична робота № 3. «Класи в C# та спадкування»**

**Мета роботи**: засвоїти основи побудови ієрархії класів на мові C#, набути практичних навичок використання прийомів об'єктно-орієнтованого програмування.

**Завдання:**

Визначити ієрархію класів (9. Перелік класів: Іграшка, Продукт, Товар, Молочний продукт).

1. Для визначення ієрархії класів потрібно зв'язати відношенням спадкоємства класів, приведені у варіанті завдання. З перерахованих класів вибрати один, який стоятиме на чолі ієрархії. Це буде абстрактний клас.

У нашому переліку класів абстрактними класом буде Товар(Item).

Лістенінг абстрактного класу Item:

abstract class Item

{

public static LinkedListNode<Item> FirstNode;

public int cost { get; set; }

public float mass { get; set; }

public float[] Size { get; set; }

public string Name { get; set; }

public string FirmName { get; set; }

public Item() { }

public Item(int cost, float mass, float[] Size, string Name, string FirmName, LinkedList<Item> list, bool add)

{

this.cost = cost;

this.mass = mass;

this.Size = Size;

this.Name = Name;

this.FirmName = FirmName;

if (add)

Add(list);

}

static public void PrintList()

{

if (FirstNode == null)

return;

Console.WriteLine("Output list content:");

for (LinkedListNode<Item> node = FirstNode; node != null; node = node.Next)

node.Value.Print();

}

public void Add(LinkedList<Item> list = null)

{

if (list != null)

{

list.AddLast(this);

FirstNode = list.First;

}

else if (FirstNode != null)

{

FirstNode.List.AddLast(this);

}

}

public virtual void Print()

{

Console.WriteLine($"Name: {Name}, FirmName: {FirmName}, Cost: {cost}, Mass: {mass}, Size: x: {Size[0]}, y: {Size[1]}, z: {Size[2]}");

}

}

1. Визначити в класах всі необхідні конструктори і деструктори.
2. Визначити в класі статичну компоненту - покажчик на початок зв'язаного списку об'єктів і статичну функцію для переглядання списку. Для додавання об'єкту в список слід передбачити метод класу, тобто об'єкт сам додає себе в список. Наприклад, ***а.Add()*** - об'єкт ***а*** додає себе в список. Включення об'єкту в список можна виконувати при створенні об'єкту, тобто помістити оператори включення в конструктор. У разі ієрархії класів, включення об'єкту в список повинно виконувати тільки конструктор базового класу. Ви повинні продемонструвати обидва ці способи.
3. Реалізувати класи.

Лістенінг класу MilkProduct:

class MilkProduct : Product

{

public float FatPercent { get; set; }

public bool Natural { get; set; }

public MilkProduct(int cost, float mass, float[] Size, string Name, string FirmName, LinkedList<Item> list, bool add, string Category, string PackageType,

bool FullAgeAcces, bool HealthHurting, DateTime CreatingDate, TimeSpan TermDuration, float FatPercent, bool Natural)

: base(cost, mass, Size, Name, FirmName, list, add, Category, PackageType, FullAgeAcces, HealthHurting, CreatingDate, TermDuration)

{

this.FatPercent = FatPercent;

this.Natural = Natural;

}

public override void Print()

{

Console.WriteLine($"Name: {Name}, FirmName: {FirmName}, Cost: {cost}, Mass: {mass}, Size: x: {Size[0]}, y: {Size[1]}, z: {Size[2]} " +

$"Fat percent: {FatPercent}, Natural: {Natural}"); //Вивід

}

}

Лістенінг класу Product:

class Product : Item

{

public string Category { get; set; }

public string PackageType { get; set; }

public bool FullAgeAcces { get; set; }

public bool HealthHurting { get; set; }

public DateTime CreatingDate { get; set; }

public TimeSpan TermDuration { get; set; }

public Product(int cost, float mass, float[] Size, string Name, string FirmName, LinkedList<Item> list, bool add, string Category, string PackageType,

bool FullAgeAcces, bool HealthHurting, DateTime CreatingDate, TimeSpan TermDuration) : base(cost, mass, Size, Name, FirmName, list, add)

{

this.Category = Category;

this.PackageType = PackageType;

this.FullAgeAcces = FullAgeAcces;

this.HealthHurting = HealthHurting;

this.CreatingDate = CreatingDate;

this.TermDuration = TermDuration;

}

public DateTime GetTermOutDate()

{

DateTime d1 = CreatingDate;

d1.AddDays(TermDuration.Days);

d1.AddDays(TermDuration.Days);

d1.AddMinutes(TermDuration.Minutes); //Реалізація терміну придатності

return d1;

}

public bool IsOutOfTerm()

{

return DateTime.Now < GetTermOutDate();

}

}

Лістенінг класу Toy:

class Toy : Item

{

public int OldTarget { get; set; }

public bool BoysToy { get; set; }

public bool PartyToy { get; set; }

public string PlayCategory { get; set; }

public Toy() : base() { }

public Toy(int cost, float mass, float[] Size, string Name, string FirmName, LinkedList<Item> list, bool add,

int OldTarget, bool BoysToy, bool PartyToy, string PlayCategory)

: base(cost, mass, Size, Name, FirmName, list, add)

{

this.OldTarget = OldTarget;

this.BoysToy = BoysToy;

this.PartyToy = PartyToy;

this.PlayCategory = PlayCategory;

}

public override void Print()

{

Console.WriteLine($"Name: {Name}, FirmName: {FirmName}, Cost: {cost}, Mass: {mass}, Size: x: {Size[0]}, y: {Size[1]}, z: {Size[2]}" +

$", Old Target: {OldTarget}, Sex: {(BoysToy ? "Male" : "Female")}, Party toy: {PartyToy}, Play Category: {PlayCategory}"); //Вивід

}

}

1. Написати демонстраційну програму, в якій створюються об'єкти різних класів і поміщаються в список, після чого список є видимим. Список є видимим шляхом виклику віртуального методу Show кожного об'єкту.

Лістенінг класу Program:

class Program

{

static void Main(string[] args)

{

LinkedList<Item> list = new LinkedList<Item>();

Console.WriteLine("Create basic list (MilkProduct)...");

MilkProduct m1 = new MilkProduct(50, 950, new float[] { 50, 50, 170 }, "GoodMilk", "AGM", list, true, "Lactic", "Solid", false, false, new DateTime(2021, 11, 6), new TimeSpan(6, 0, 0), 2.5F, false);

MilkProduct m2 = new MilkProduct(55, 950, new float[] { 200, 20, 40 }, "MITII", "SVE", list, true, "Lactic", "Solid", false, false, new DateTime(2021, 11, 7), new TimeSpan(4, 0, 0), 3.2F, false);

MilkProduct m3 = new MilkProduct(105, 1890, new float[] { 65, 40, 200 }, "LargeMamy", "AGN", list, false, "Lactic", "Membrance", false, false, new DateTime(2021, 10, 26), new TimeSpan(10, 0, 0), 1F, true);

MilkProduct m4 = new MilkProduct(28, 450, new float[] { 80, 80, 55 }, "MinyGO", "FLKS", list, false, "Lactic", "Membrance", true, false, new DateTime(2021, 11, 11), new TimeSpan(8, 0, 0), 1F, false);

Console.WriteLine("Done!\n\n");

m3.Add();

m4.Add();

Item.PrintList();

Console.WriteLine("\n\nAdd (Toy) instances...");

Toy t1 = new Toy(270, 105, new float[] { 220, 110, 80 }, "LOOTGun", "IBM", list, false, 10, true, true, "Dynamic, Shooter");

Toy t2 = new Toy(150, 20, new float[] { 25, 15, 5 }, "PheraCar", "Hoot", list, true, 6, true, false, "Strategy, Imagination");

Toy t3 = new Toy(370, 120, new float[] { 50, 30, 110 }, "PrettyDoll", "MMS", list, true, 6, false, false, "Strategy, Imagination");

Toy t4 = new Toy(200, 200, new float[] { 1100, 1100, 50 }, "MonopolyS+", "KNO", list, false, 10, true, true, "Strategy, Tactics");

Console.WriteLine("Done!\n\n");

t1.Add();

t4.Add();

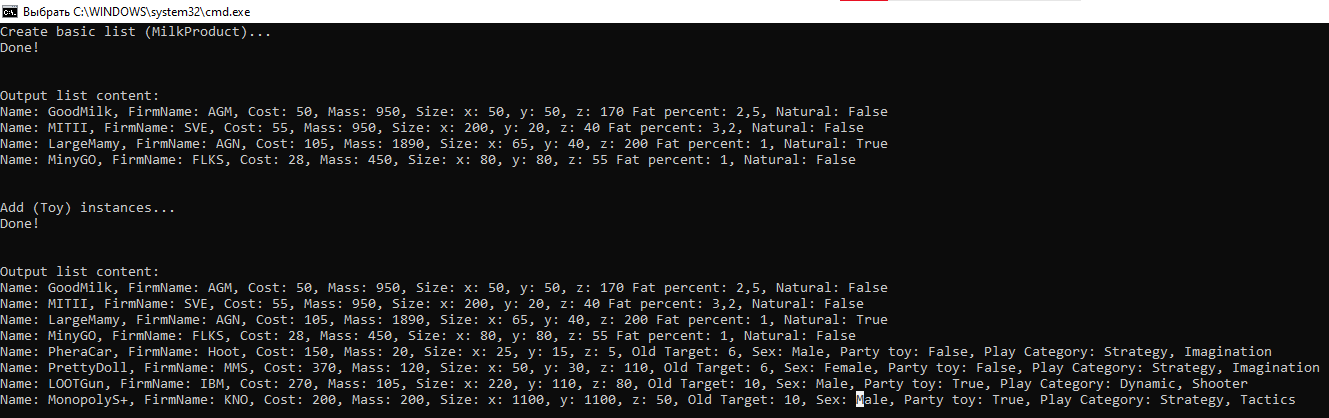
Item.PrintList();

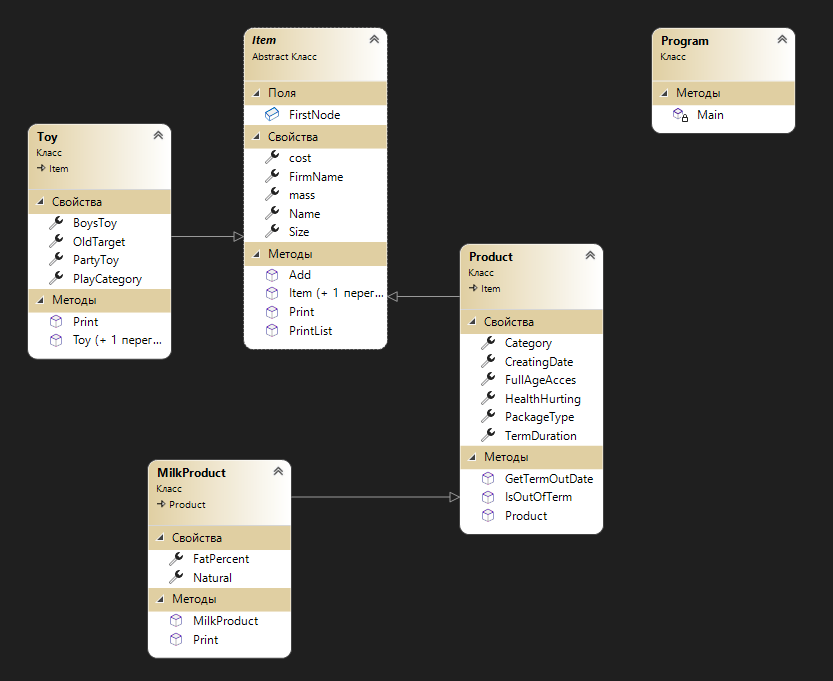
Console.ReadLine();

}

}

**Демонстрація роботи програми**



**UML-діаграма класів проекту**

**Практична робота № 4. «Обробка виключень в мові C#»**

**Мета роботи**: засвоїти основи обробки виключень в мові C#, набути практичних навичок обробки виключень.

**Завдання**. Розробити простий калькулятор, ввівши в нього обробку виключень, пов'язаних з неправильним введенням даних.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

class WrongOperator : Exception

{

public WrongOperator(string oper) : base("Wrong opertor pass to calculate")

{

Console.WriteLine($"Not a math operator: [{oper}]");

}

}

Виключення що виникає при неправильному написі математичної операції. Тобто все що не є в прикладі.

class WrongOperands : Exception

{

public WrongOperands(string operands) : base("Wrong operands pass to calculate")

{

Console.WriteLine($"Not a value to calculate: [{operands}]");

}

}

Виключення, що виникає при неправильно введеному знаку цифри.

class TryDivideByZero : Exception

{

public TryDivideByZero() : base()

{

Console.WriteLine("Can't divide by zero");

}

}

Виключення, що виникає при спробі поділити на нуль.

static float Calculate(float f1, float f2, string oper)

{

switch (oper)

{

case "+": return f1 + f2;

case "-": return f1 - f2;

case "\*": return f1 \* f2;

case "/": if (f2 == 0) throw new TryDivideByZero(); return f1 / f2;

case "^": return (float)Math.Pow(f1, f2);

default: throw new WrongOperator(oper);

}

}

Метод Calculate, який реалізує прості можливості калькулятора. При випадку коли операції в наборі значень switch, викликається виключення WrongOperator

static void Main(string[] args)

{

bool faile = false;

Console.WriteLine("Simple calculator load...");

Console.Write("First you write ");

Console.ForegroundColor = ConsoleColor.Green;

Console.Write("left operand");

Console.ForegroundColor = ConsoleColor.White;

Console.Write(", after, ");

Console.ForegroundColor = ConsoleColor.Red;

Console.Write("operation ");

Console.ForegroundColor = ConsoleColor.White;

Console.Write("to calculate, and then ");

Console.ForegroundColor = ConsoleColor.Green;

Console.Write("right operand\n");

Console.ForegroundColor = ConsoleColor.White;

Console.WriteLine("Example: 2 + 2\n");

Реалізація елементарного інтерфейсу через використання кольорів.

do

{

try

{

faile = false;

Console.ForegroundColor = ConsoleColor.Green;

Console.Write("left operand (float): ");

Console.ForegroundColor = ConsoleColor.White;

var left\_out = Console.ReadLine();

if (!float.TryParse(left\_out, out float left))

throw new WrongOperands(left\_out);

Конструкція, що перевіряє перший операнд, яку вводить користувач, через функцію TryParse, якщо це не число то викликається виключення WrongOperands. Також виводить напис зеленого кольору, який пропонує ввести число.

Console.ForegroundColor = ConsoleColor.Red;

Console.Write("math operation (+,-,\*,/,^): ");

Console.ForegroundColor = ConsoleColor.White;

var oper = Console.ReadLine();

Console.ForegroundColor = ConsoleColor.Green;

Console.Write("right operand (float): ");

Console.ForegroundColor = ConsoleColor.White;

var right\_out = Console.ReadLine();

if (!float.TryParse(right\_out, out float right))

throw new WrongOperands(right\_out);

Ще дві конструкцію, які викликають різнокольорові написи, що пропонуються написати математичну операцію або другий операнд для операції.

Console.WriteLine(Calculate(left, right, oper)); //Викликання методу Calculate, з деякими перевіреними даними(крім математичної операції, вона перевіряється у самому методі Calculate()), що ввів користувач на коректність.

}

catch (Exception)

{

faile = true;

Console.WriteLine();

}

} while (faile);

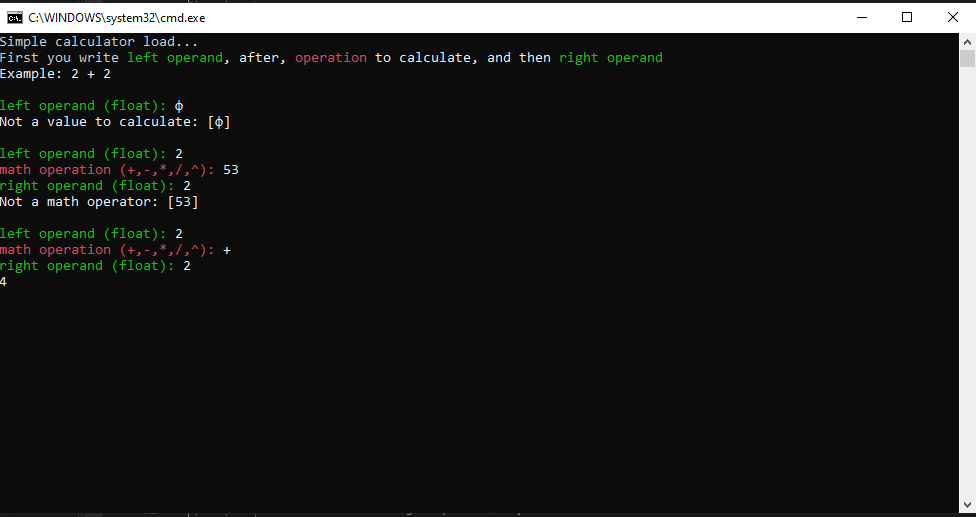
Console.ReadLine();

}

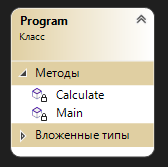
}

}

**Демонстрація роботи програми**



**UML-діаграма класів проекту**



**Практична робота № 5 «Абстрактні класи»**

**Тема:** Абстрактні класи.

**Мета роботи**: Ознайомитися з використанням абстрактних і безплідних(sealed) класів, як із реалізацією принципу поліморфізму мовою С#.

Завдання 1:Розробити програму, що реалізує роботу із класами.

Завдання 2:Створити проект Visual Studio 2019 для програмованого завдання:

* **Завдання 2.1:** Визначити абстрактний ***клас Shape***з абстрактними властивостями та методами.
* **Завдання 2.2:** Визначити абстрактний клас ***Shape*** з абстрактними властивостями та методами.
* **Завдання 2.3:** Визначити абстрактний клас ***Triangle*** з абстрактними властивостями та методами

Повний лістинг програми :

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace Test

{

interface IPrint

{

void Print();

}

abstract class Shape : IPrint

{

public string ShapeName { get; set; }

public abstract int VertexAmount { get; }

public abstract float GetArea();

public void Print()

{

Console.WriteLine(ToString());

}

}

class Rectangle : Shape

{

public Rectangle(float width, float height)

{

Width = width;

Height = height;

ShapeName = "Прямокутник";

}

public float Width { get; set; }

public float Height { get; set; }

public override int VertexAmount => 4;

public override float GetArea() => Width \* Height;

public override string ToString()

{

return $"Фігура {ShapeName}, ширина : {Width}, Висота : {Height}, Площа : {GetArea()}";

}

}

class Square : Rectangle

{

public Square(float side) : base(side, side)

{

ShapeName = "Квадрат";

}

public override string ToString()

{

return $"Фігура {ShapeName}, довжина сторони : {Width}, Площа : {GetArea()}";

}

}

class Circuit : Shape

{

public float Radius { get; set; }

public Circuit(float radius)

{

Radius = radius;

ShapeName = "Коло";

}

public override float GetArea() => (float)Math.PI \* Radius \* Radius;

public override int VertexAmount => 0;

public override string ToString()

{

return $"Фігура {ShapeName}, радіус : {Radius}, Площа : {GetArea()}";

}

}

abstract class Triangle : Shape

{

public float SideA { get; set; }

public float SideB { get; set; }

public float Angle { get; set; }

protected float RadAngle { get => Angle \* (float)Math.PI / 180; }

public virtual float SideC { get => (float)Math.Sqrt( SideA \* SideA + SideB \* SideB - 2 \* SideA \* SideB \* Math.Cos(RadAngle));}

public Triangle(float sideA, float sideB, float angle)

{

SideA = sideA;

SideB = sideB;

Angle = angle;

}

public override float GetArea() => SideA \* SideB \* (float)Math.Sin(RadAngle) / 2;

public abstract float GetPerimeter();

public override int VertexAmount => 3;

}

class EqTriangle : Triangle

{

public EqTriangle(float side) : base(side, side , 60)

{

ShapeName = "Рівносторонній трикутник";

}

public override int VertexAmount => 3;

public override float SideC => SideA;

public override float GetPerimeter() => SideA \* 3;

public override string ToString()

{

return $"Фігура {ShapeName}, довжина сторони : {SideA}, Площа : {GetArea()}";

}

}

class IsTriangle : Triangle

{

public IsTriangle(float side, float angle) : base(side, side, angle)

{

ShapeName = "Рівнобедрений трикутник";

}

public override float GetPerimeter() => SideA + SideB + SideC;

public override string ToString()

{

return $"Фігура {ShapeName}, довжина 1 і 2 сторони : {SideA}, кут між 1 і 2 стороною : {Angle} Площа : {GetArea()}";

}

}

class RectTriangle : Triangle

{

public RectTriangle(float sideA, float sideB): base(sideA, sideB, 90)

{

ShapeName = "Прямокутний трикутник";

}

public override float SideC => (float)Math.Sqrt(SideA\*SideA + SideB\*SideB);

public override float GetPerimeter() => SideA + SideB + SideC;

public override string ToString()

{

return $"Фігура {ShapeName}, довжина 1 сторони : {SideA}, довжина 2 сторони: {SideB} Площа : {GetArea()}";

}

}

class Pentagon : Shape

{

public float Side { get; set; }

public override int VertexAmount => 5;

public Pentagon(float side)

{

Side = side;

ShapeName = "Правильний п'ятикутник";

}

public override float GetArea()

{

return (Side \* Side / 4) \* (float)Math.Sqrt(25 + 10.0 \* Math.Sqrt(5.0));

}

public override string ToString()

{

return $"Фігура {ShapeName}, довжина сторони : {Side}, Площа : {GetArea()}";

}

}

class Ellipse : Shape

{

public float A { get; set; }

public float B { get; set; }

public override int VertexAmount => 0;

public Ellipse(float a, float b)

{

A = a;

B = b;

ShapeName = "Еліпс";

}

public override float GetArea()

{

return (float)Math.PI \* A \* B;

}

}

class Picture

{

Shape[] shpArray;

private int n = 1;

public int N { get => n; set { if (value <= 0) return; n = value; shpArray = new Shape[n]; } }

public Picture(int arraySize)

{

N = arraySize;

}

public void FillArray(params Shape[] shapes)

{

for (int i = 0; i < shapes.Length && i < n; i++)

{

shpArray[i] = shapes[i];

}

}

public void PrintArray()

{

foreach (var item in shpArray)

{

if(item != null)

item.Print();

}

}

}

class Program

{

static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.Unicode;

Console.InputEncoding = System.Text.Encoding.Unicode;

Circuit circuit = new Circuit(5);

Square square = new Square(5);

Pentagon pentagon = new Pentagon(5);

Picture picture = new Picture(3);

Console.WriteLine("Заповнення масиву...\n");

picture.FillArray(circuit, square, pentagon);

Console.WriteLine("\nВивід масиву фігур : \n");

picture.PrintArray();

Picture picture2 = new Picture(3);

Console.WriteLine("\nЗаповнення масиву трикутників...\n");

Triangle eqTriangle = new EqTriangle(3);

Triangle isTriangle = new IsTriangle(3, 45);

Triangle rectTriangle = new RectTriangle(3, 5);

picture2.FillArray(eqTriangle, isTriangle, rectTriangle);

Console.WriteLine("\nВивід масиву трикутників : \n");

picture2.PrintArray();

Console.ReadLine();

}

}

}

Визначено інтерфейс IPrint з методом Print, який реалізує абстрактний клас Shape, в якому є поля з назвою фігури, абстрактна властивість кількість вершин, абстрактний метод GetArea, та метод Print, який виводить в консоль інформацію про фігуру.

interface IPrint

{

void Print();

}

abstract class Shape : IPrint

{

public string ShapeName { get; set; }

public abstract int VertexAmount { get; }

public abstract float GetArea();

public void Print()

{

Console.WriteLine(ToString());

}

}

Далі йдуть класи Rectangle, Circuit, Pentagon, Ellipse – які наслідують клас Shape, і реалізують властивість VertexAmount і метод GetArea згідно до контексту фігури. Також в їх конструкторах ініціалізується поле з іменем фігури.

class Rectangle : Shape

{

public Rectangle(float width, float height)

{

Width = width;

Height = height;

ShapeName = "Прямокутник";

}

public float Width { get; set; }

public float Height { get; set; }

public override int VertexAmount => 4;

public override float GetArea() => Width \* Height;

public override string ToString()

{

return $"Фігура {ShapeName}, ширина : {Width}, Висота : {Height}, Площа : {GetArea()}";

}

}

class Circuit : Shape

{

public float Radius { get; set; }

public Circuit(float radius)

{

Radius = radius;

ShapeName = "Коло";

}

public override float GetArea() => (float)Math.PI \* Radius \* Radius;

public override int VertexAmount => 0;

public override string ToString()

{

return $"Фігура {ShapeName}, радіус : {Radius}, Площа : {GetArea()}";

}

}

class Pentagon : Shape

{

public float Side { get; set; }

public override int VertexAmount => 5;

public Pentagon(float side)

{

Side = side;

ShapeName = "Правильний п'ятикутник";

}

public override float GetArea()

{

return (Side \* Side / 4) \* (float)Math.Sqrt(25 + 10.0 \* Math.Sqrt(5.0));

}

public override string ToString()

{

return $"Фігура {ShapeName}, довжина сторони : {Side}, Площа : {GetArea()}";

}

}

class Ellipse : Shape

{

public float A { get; set; }

public float B { get; set; }

public override int VertexAmount => 0;

public Ellipse(float a, float b)

{

A = a;

B = b;

ShapeName = "Еліпс";

}

public override float GetArea()

{

return (float)Math.PI \* A \* B;

}

}

Є також класи EqTriangle, IsTriangle, RectTriangle – які наслідують абстрактний клас Triangle , які мають властивості зі сторонами і кути.

abstract class Triangle : Shape

{

public float SideA { get; set; }

public float SideB { get; set; }

public float Angle { get; set; }

protected float RadAngle { get => Angle \* (float)Math.PI / 180; }

public virtual float SideC { get => (float)Math.Sqrt( SideA \* SideA + SideB \* SideB - 2 \* SideA \* SideB \* Math.Cos(RadAngle));}

public Triangle(float sideA, float sideB, float angle)

{

SideA = sideA;

SideB = sideB;

Angle = angle;

}

public override float GetArea() => SideA \* SideB \* (float)Math.Sin(RadAngle) / 2;

public abstract float GetPerimeter();

public override int VertexAmount => 3;

}

class EqTriangle : Triangle

{

public EqTriangle(float side) : base(side, side , 60)

{

ShapeName = "Рівносторонній трикутник";

}

public override int VertexAmount => 3;

public override float SideC => SideA;

public override float GetPerimeter() => SideA \* 3;

public override string ToString()

{

return $"Фігура {ShapeName}, довжина сторони : {SideA}, Площа : {GetArea()}";

}

}

class IsTriangle : Triangle

{

public IsTriangle(float side, float angle) : base(side, side, angle)

{

ShapeName = "Рівнобедрений трикутник";

}

public override float GetPerimeter() => SideA + SideB + SideC;

public override string ToString()

{

return $"Фігура {ShapeName}, довжина 1 і 2 сторони : {SideA}, кут між 1 і 2 стороною : {Angle} Площа : {GetArea()}";

}

}

class RectTriangle : Triangle

{

public RectTriangle(float sideA, float sideB): base(sideA, sideB, 90)

{

ShapeName = "Прямокутний трикутник";

}

public override float SideC => (float)Math.Sqrt(SideA\*SideA + SideB\*SideB);

public override float GetPerimeter() => SideA + SideB + SideC;

public override string ToString()

{

return $"Фігура {ShapeName}, довжина 1 сторони : {SideA}, довжина 2 сторони: {SideB} Площа : {GetArea()}";

}

}

Клас Square наслідує Rectangle та перевизначає метод ToString()

class Square : Rectangle

{

public Square(float side) : base(side, side)

{

ShapeName = "Квадрат";

}

public override string ToString()

{

return $"Фігура {ShapeName}, довжина сторони : {Width}, Площа : {GetArea()}";

}

}

В класі Picture є масив з класами Shape і метод FillArray, для його заповнення. Метод PrintArray виводить інформації про всі фігури в масиві.

class Picture

{

Shape[] shpArray;

private int n = 1;

public int N { get => n; set { if (value <= 0) return; n = value; shpArray = new Shape[n]; } }

public Picture(int arraySize)

{

N = arraySize;

}

public void FillArray(params Shape[] shapes)

{

for (int i = 0; i < shapes.Length && i < n; i++)

{

shpArray[i] = shapes[i];

}

}

public void PrintArray()

{

foreach (var item in shpArray)

{

if(item != null)

item.Print();

}

}

}

В методі Main написана демонстрація виводу інформації про фігури з масиву в класі Picture (Рисунок 5.1)

class Program

{

static void Main(string[] args)

{

Console.OutputEncoding = System.Text.Encoding.Unicode;

Console.InputEncoding = System.Text.Encoding.Unicode;

Circuit circuit = new Circuit(5);

Square square = new Square(5);

Pentagon pentagon = new Pentagon(5);

Picture picture = new Picture(3);

Console.WriteLine("Заповнення масиву...\n");

picture.FillArray(circuit, square, pentagon);

Console.WriteLine("\nВивід масиву фігур : \n");

picture.PrintArray();

Picture picture2 = new Picture(3);

Console.WriteLine("\nЗаповнення масиву трикутників...\n");

Triangle eqTriangle = new EqTriangle(3);

Triangle isTriangle = new IsTriangle(3, 45);

Triangle rectTriangle = new RectTriangle(3, 5);

picture2.FillArray(eqTriangle, isTriangle, rectTriangle);

Console.WriteLine("\nВивід масиву трикутників : \n");

picture2.PrintArray();

Console.ReadLine();

}

}

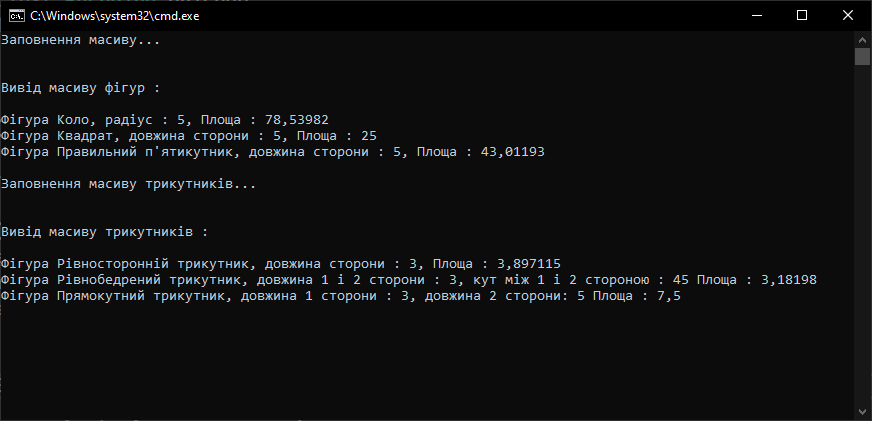


Рисунок 5.1 – Демонстрація роботи програми

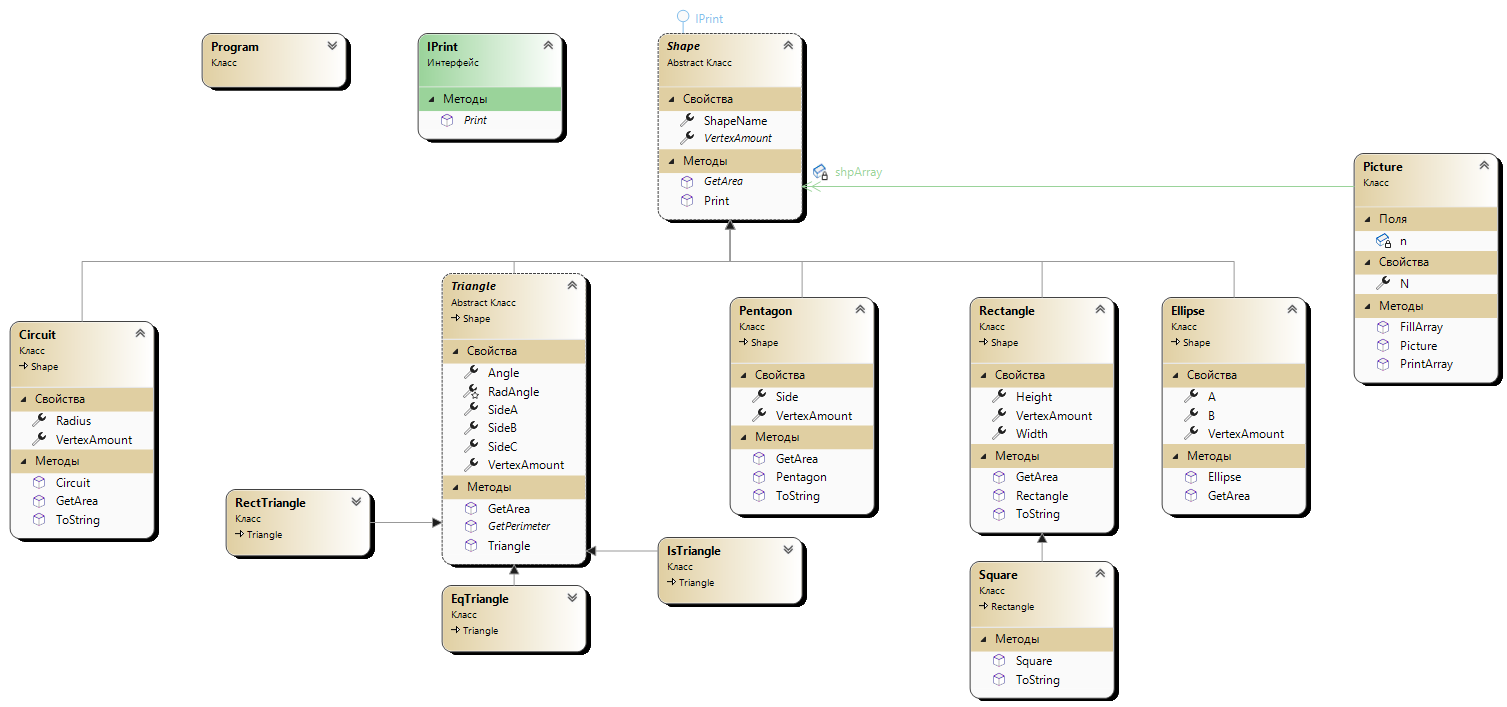


Рисунок 5.2 – UML діаграма класів

**Практична робота №6. «Віртуальні методи»**

**Мета роботи**: Навчитися застосовувати віртуальні класи при розробці

проектів мовою С#.

**Завдання**

1) Організувати обчислення сумарної величини доходу працівників

та максимальне для менеджерів.

У класі base\_worker створений список List<base\_worker>, визначено конструктор, який ініціалізує поля імені, віку, доходу, та значень списку. Також визначено метод DoStuff, який виводить повідомлення в консоль:

class base\_worker

{

public List<base\_worker> list;

public string name;

public int old;

public float salary;

public base\_worker(string name, int old, float salary, ref List<base\_worker> list)

{

this.name = name;

this.old = old;

this.salary = salary;

this.list = list;

list.Add(this);

}

public virtual void DoStuff()

{ Console.WriteLine("Called \"DoStuff\" in base\_worker"); }

}

Створено клас worker, який наслідує base\_worker, в ньому конструктор, який передає параметри базовому класу. Визначено метод DoStuff для підрахунку та відображення сумарного доходу всіх працівників.

class worker : base\_worker

{

public worker(string name, int old, float salary, ref List<base\_worker> list)

: base(name, old, salary, ref list)

{ }

public override void DoStuff()

{

base.DoStuff();

float sum = 0;

list.ForEach((a) => { sum += a.salary; });

Console.WriteLine($"Summary salary of all worker: {sum}");

}

}

Клас Manager наслідує base\_worker, в ньому визначено конструктор. Визначено метод DoStuff для підрахунку та відображення максимального доходу всіх менеджерів.

class Manager : base\_worker

{

public Manager(string name, int old, float salary, ref List<base\_worker> list)

: base(name, old, salary, ref list) {}

public override void DoStuff()

{

base.DoStuff();

float max = 0;

for (int i = 0; i < list.Count; i++)

if (list[i].salary > max)

max = list[i].salary;

Console.WriteLine($"Max salary of all managers: {max}");

}

}

Лістинг класу Program:

В методі main створюються списки, які заповнюються робітниками та менеджерами та за допомогою методу DoStuff у консоль виводяться підрахунки.

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Create workers and managers...");

List<base\_worker> list\_w= new List<base\_worker>();

worker w1 = new worker("Koon", 23, 2599, ref list\_w);

worker w2 = new worker("Mon", 99, 1500, ref list\_w);

worker w3 = new worker("Jon", 53, 100, ref list\_w);

List<base\_worker> list\_m = new List<base\_worker>();

Manager m1 = new Manager("Oldi", 16, 195, ref list\_m);

Manager m2 = new Manager("Miir", 29, 1599, ref list\_m);

Manager m3 = new Manager("Bob", 30, 455, ref list\_m);

Console.WriteLine("Done!\n\n");

Console.WriteLine("Call in variable {w1} fuction -> DoStuff() (Worker)");

w1.DoStuff();

Console.WriteLine("\n\nCall in variable {m1} fuction -> DoStuff() (Manager)");

m1.DoStuff();

Console.ReadLine();

}

}

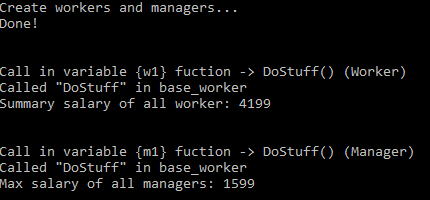


Рисунок 1 – демонстрація роботи програми

**UML діаграми класів**

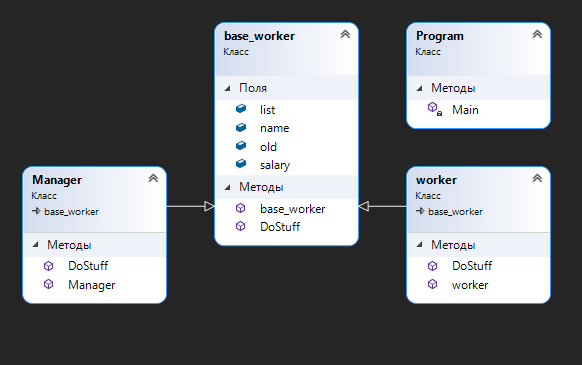


Рисунок 2 – UML діаграма класів

**Практична робота № 7. «Шаблони функцій та параметризовані**

**підпрограми сортування»**

**Мета роботи**: Навчитися застосовувати різні шаблони функцій для

вирішення задач при програмуванні мові С#.

**Завдання**

1) Написати та протестувати параметризовану підпрограму

сортування методом, що має звичайні та універсальні параметри. Відладити

її на цілих числах і числах з плаваючою точкою.

2) Визначити клас об`єктів масиву, що призначений для сортування.

Перевантажити йому операцію привласнення та операції порівняння >, <=, ==, >=, >=, >.

3) Написати та протестувати програму, яка сортує масив об`єктів

побудованого класу за допомогою написаної параметризованої підпрограми.

Клас Bus реалізує інтерфейс IComparable, класі створена змінна double mass, яка відповідає за масу автобусу. Далі метод CompareTo порівнює об’єкти по масі. Перевантажені оператори <, >, <=, >= викликають метод CompareTo, згідно з його результатом виконання повертають булеве значення.

class Bus : IComparable

{

public double mass;

public override string ToString()

{

return mass.ToString();

}

public Bus(double mass)

{

this.mass = mass;

}

public int CompareTo(object obj)

{

return mass.CompareTo(((Bus)obj).mass);

}

public static bool operator >(Bus b1, Bus b2)

{

return b1.CompareTo(b2) == 1;

}

public static bool operator <(Bus b1, Bus b2)

{

return b1.CompareTo(b2) == -1;

}

public static bool operator >=(Bus b1, Bus b2)

{

return b1.CompareTo(b2) == 1 || b1.CompareTo(b2) == 0;

}

public static bool operator <=(Bus b1, Bus b2)

{

return b1.CompareTo(b2) == -1 || b1.CompareTo(b2) == 0;

}

Перевантажений метод Equals порівнює об’єкти bus і порівнює об’єкти по масі.

public override bool Equals(object obj)

{

if ((obj == null) || !this.GetType().Equals(obj.GetType()))

{

return false;

}

return CompareTo(obj) == 0;

}

public override int GetHashCode()

{

return base.GetHashCode();

}

Перевантажений оператор порівняння повертає результат виконання методу Equals(b2)

public static bool operator ==(Bus b1, Bus b2)

{

return b1.Equals(b2);

}

public static bool operator !=(Bus b1, Bus b2)

{

return !b1.Equals(b2);

}

}

Метод Sort приймає параметри із списком з інтами, булева змінна asc потрібна для зміни напрямку сортування, далі йде сортування методом бульбашки. В результаті повертає відсортований список.

public static List<int> Sort(List<int> list, bool asc = true)

{

bool change = false;

do

{

change = false;

for (int i = 0; i < list.Count - 1; i++)

{

Predicate<int> compare = asc ?

(left) => list[i] > left :

(left) => list[i] < left;

if (compare(list[i + 1]))

{

int a = list[i];

list[i] = list[i + 1];

list[i + 1] = a;

change = true;

}

}

} while (change);

return list;

}

Те ж саме сортування, але призначене для універсальних параметрів, які реалізують інтерфейс IComparable, повертає відсортований список:

public static List<T> Sort<T>(List<T> list, bool asc = true) where T : IComparable

{

bool change = false;

do

{

change = false;

for (int i = 0; i < list.Count - 1; i++)

{

bool ifi = (Convert.ToInt32(asc) \* 2 - 1) \* list[i].CompareTo(list[i + 1]) > 0;

if (ifi)

{

T a = list[i];

list[i] = list[i + 1];

list[i + 1] = a;

change = true;

}

}

} while (change);

return list;

}

Визначено метод PrintList, який визначено для виводу списку у консолі.

public static void PrintList<T>(List<T> l)

{

Console.WriteLine();

l.ForEach( (a) => Console.Write(a.ToString()+" "));

Console.WriteLine();

}

Далі демонстрація сортування списків різних типів.

static void Main(string[] args)

{

Console.WriteLine("Initialize array's...");

List<int> num = new List<int>() { 2, 3, 423, -1, 14 };

List<string> strings = new List<string>() { "ab", "aa", "1c" };

List<double> f = new List<double>() { 1.1, -7.8, 99.99, 13.45 };

List<Bus> b = new List<Bus>() { new Bus(15.3), new Bus(10), new Bus(12), new Bus(17) };

Console.WriteLine("Done!\n\n");

Console.Write("num array (int):");

PrintList<int>(num);

Console.Write("strings array (string):");

PrintList<string>(strings);

Console.Write("f array (double):");

PrintList<double>(f);

Console.Write("b array (Bus):");

PrintList<Bus>(b);

Console.WriteLine("\n\nStart sort array's...");

num = Sort(num);

strings = Sort<string>(strings);

f = Sort<double>(f, false);

b = Sort<Bus>(b);

Console.WriteLine("Done!\n\n");

Console.Write("num array (int):");

PrintList<int>(num);

Console.Write("strings array (string):");

PrintList<string>(strings);

Console.Write("f array (double):");

PrintList<double>(f);

Console.Write("b array (Bus):");

PrintList<Bus>(b);

Console.ReadLine();

}

}

}

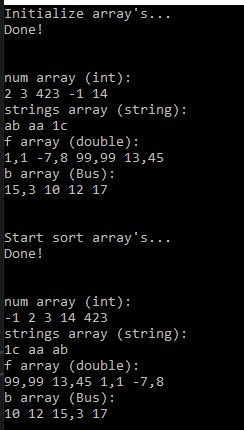
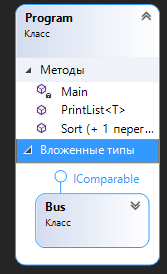


Рисунок 1 – демонстрація роботи програми

UML діаграма класів



**Практична робота № 8. «Інтерфейси»**

**Мета роботи:** засвоїти основи створення класів в мові C#, набути

практичних навичок застосування принципів поліморфізму об’єктно-

орієнтованого програмування.

Варіант 9

1. Побудувати ієрархію класів у відповідності до варіанта завдання. Розширити дану ієрархію класів з використанням абстрактного класу в якості основи ієрархії. Продемонструвати приклад використання поліморфізму методів.
2. Реалізувати для ієрархії механізм інтерфейсів, при цьому один з класів повинен реалізовувати як мінімум два інтерфейси. Використовувати для перевірки всіх методів багатоадресний делегат.

Перед тим як демонструвати код для практичної роботи, потрібно описати розроблене рішення графічного інтерфейсу користувача.

На рисунку 1 зображено вигляд вікна програми під час першого запуску.

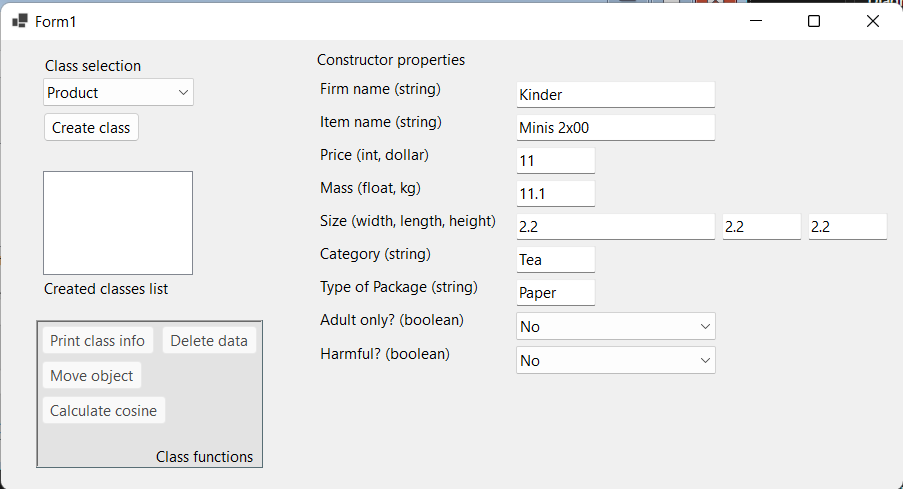


Рисунок 1 – Вигляд вікна програми

Опис елементів вікна

Для того щоб створити екземпляр класу, потрібно обрати який саме. Для цього в елементі choose\_class\_combo (ComboBox) обирається один з класів (Рис. 2)

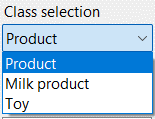


Рисунок 2 – Вигляд елемента вікна choose\_class\_combo

Після, потрібно заповнити поля конструктору класу. На вікні розташований елемент constructor\_class\_table (TableLayoutPanel), який кожен при зміні індексу елемента choose\_class\_combo генерує обов’язкові поля (Рисунок 3).

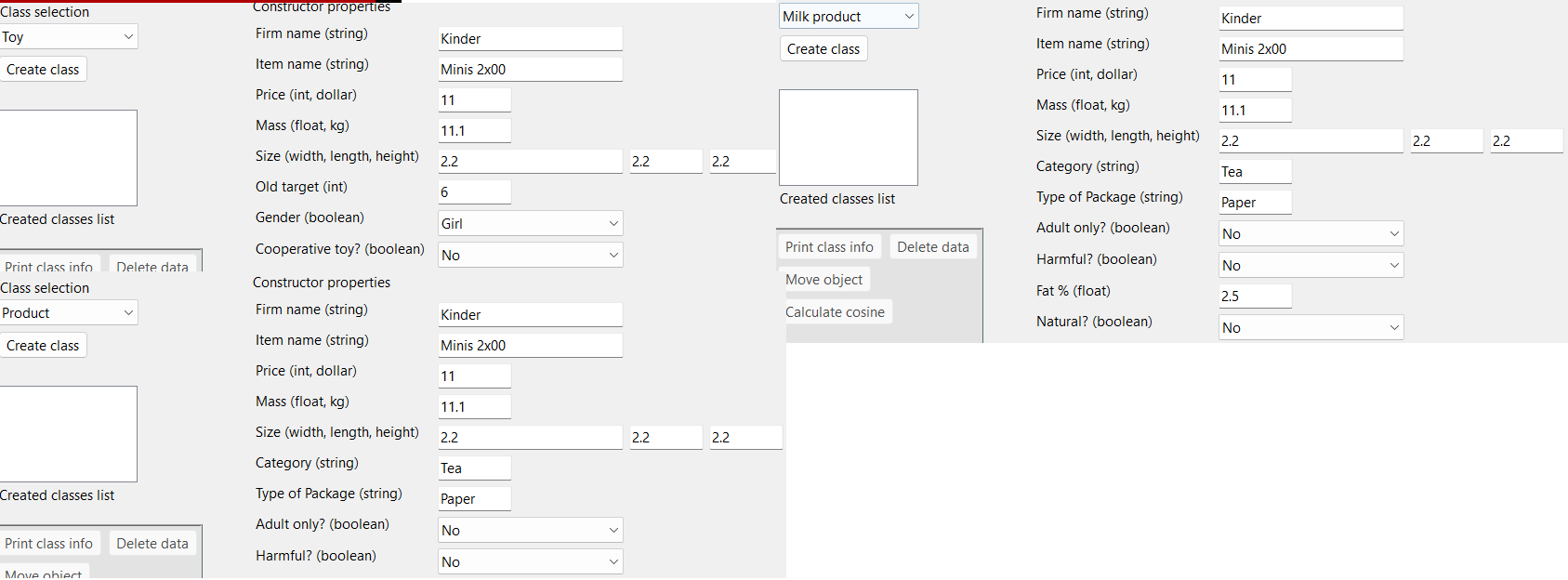


Рисунок 3 – Вигляд елементу вікна constructor\_class\_table

Після заповнення полів конструктора класу об’єкт створюється і додається до списку створених єкземплярів. На формі це елемент class\_list (ListBox) (Рис 4)

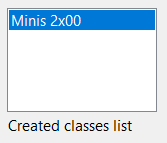


Рисунок 4 – Вигляд елементу вікна class\_list

Якщо обрати один елемент з списку, то внизу форми активуються кнопки, для виклику методів відповідного класу (Рис 5).

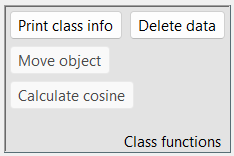


Рисунок 5 – Вигляд елементів вікна для виклику методів

Побудувати ієрархію класів у відповідності до варіанта завдання. Розширити дану ієрархію класів з використанням абстрактного класу в якості основи ієрархії.

public abstract class item:IComparable<item>, ISay

||

class Toy : item, IMove

&&

class product : item, IErase

||

class milk\_product : product, IMath

Продемонструвати приклад використання поліморфізму методів.

В абстрактному класі abstract class item:IComparable<item>, ISay є абстрактний метод ->

abstract public void Say();

Його реалізація в інших класах ->

Product:

public override void Say()

{

MessageBox.Show($"Firm name: {FirmName}, Name: {Name}, " +

$"Price: {cost}, Mass: {mass}, Size: x {Size[0]}, y {Size[1]}, z {Size[2]}, " +

$"Category: {Category}, Package type: {PackageType}, 18+: {FullAgeAcces}, " +

$"Health hurting: {HealthHurting}", "Print fuction", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

MilkProduct:

public override void Say()

{

MessageBox.Show($"Firm name: {FirmName}, Name: {Name}, " +

$"Price: {cost}, Mass: {mass}, Size: x {Size[0]}, y {Size[1]}, z {Size[2]}, " +

$"Category: {Category}, Package type: {PackageType}, 18+: {FullAgeAcces}, " +

$"Health hurting: {HealthHurting}, " +

$"Fat percent: {FatPercent}, Natural: {Natural}", "Print fuction", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

Toy:

public override void Say()

{

MessageBox.Show($"Firm name: {FirmName}, Name: {Name}, " +

$"Price: {cost}, Mass: {mass}, Size: x {Size[0]}, y {Size[1]}, z {Size[2]}, " +

$"Old target: {OldTarget}, Sex: {(BoysToy? "Male": "Female")}, " +

$"Party toy: {PartyToy}", "Print fuction", MessageBoxButtons.OK, MessageBoxIcon.Information);

1. Інтерфейси, які використовуються:

interface ISay

{

void Say();

}

interface IErase

{

void EraseMy();

}

interface IMove

{

static int velocity = 0;

static int location = 0;

void Move();

}

interface IMath

{

float GetCosine();

}

IComparable<item>

Іх реалізація в коді:

class product : item, IErase

public virtual void EraseMy()

{

cost = 0;

mass = 0;

Size = new float[3]{ 0, 0, 0};

Name = "";

FirmName = "";

Category = "";

PackageType = "";

FullAgeAcces = false;

HealthHurting = false;

}

class milk\_product : product, IMath

public float GetCosine()

{

if (Convert.ToInt32(Natural) == 0)

throw new DivideByZeroException();

float res = (float)Math.Cos(FatPercent/Convert.ToInt32(Natural));

MessageBox.Show($"Cosine of {FatPercent + Convert.ToInt32(Natural)}: {res}", "Cosine function", MessageBoxButtons.OK, MessageBoxIcon.Information);

return res;

}

public override void EraseMy()

{

base.EraseMy();

FatPercent = 0;

Natural = false;

}

class Toy : item, IMove

public void Move()

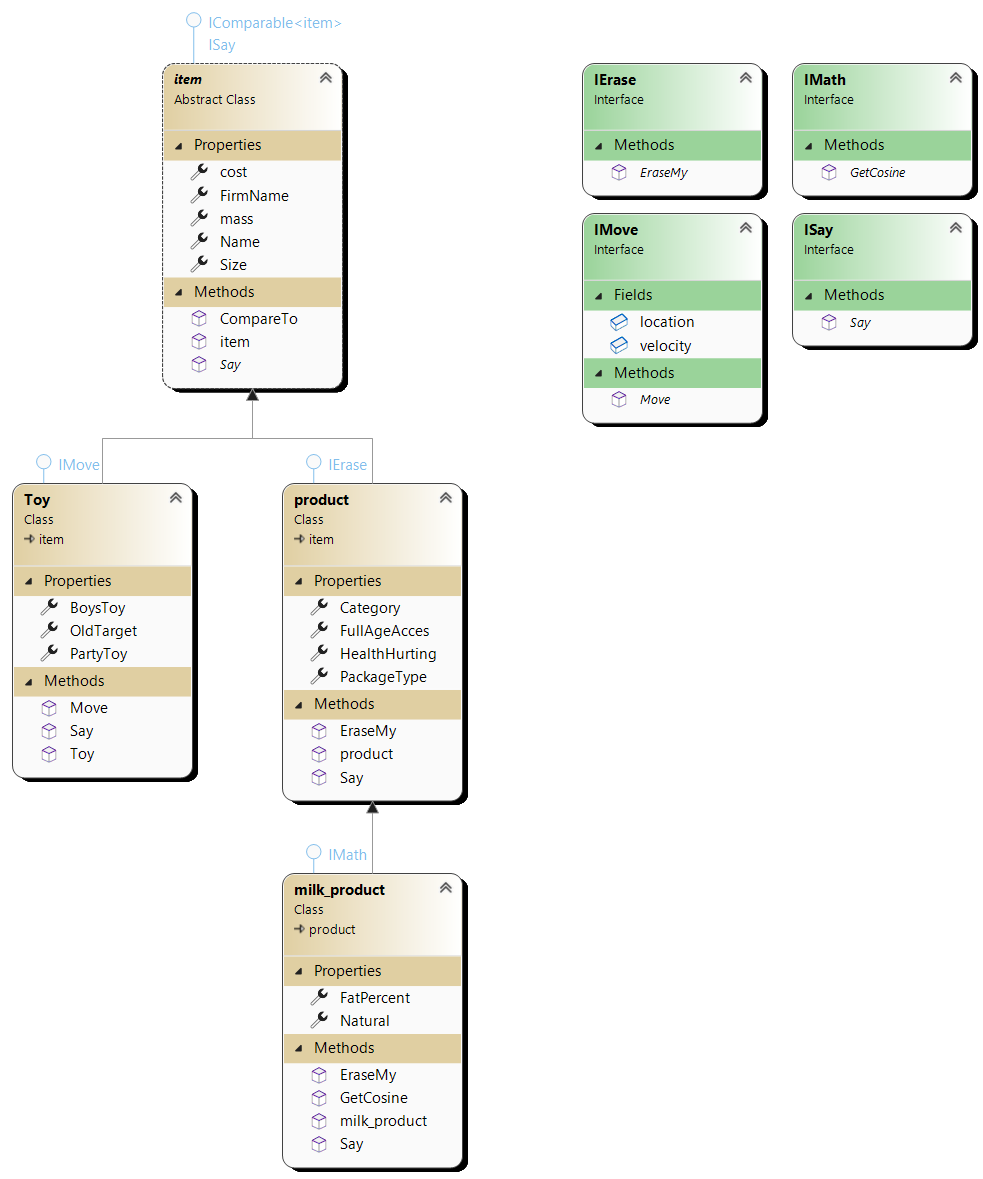
{

IMove.location += IMove.velocity;

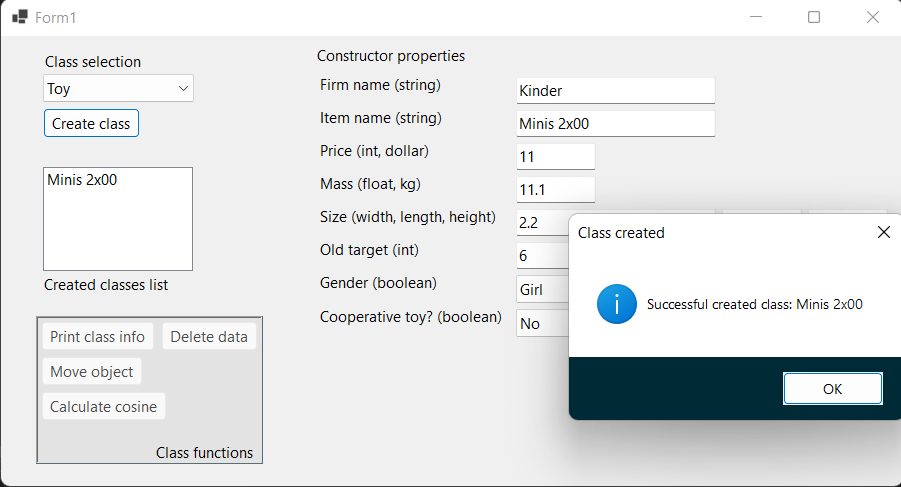
MessageBox.Show($"My velocity: {IMove.velocity}, my location: {IMove.location}", "Move fuction", MessageBoxButtons.OK, MessageBoxIcon.Information);

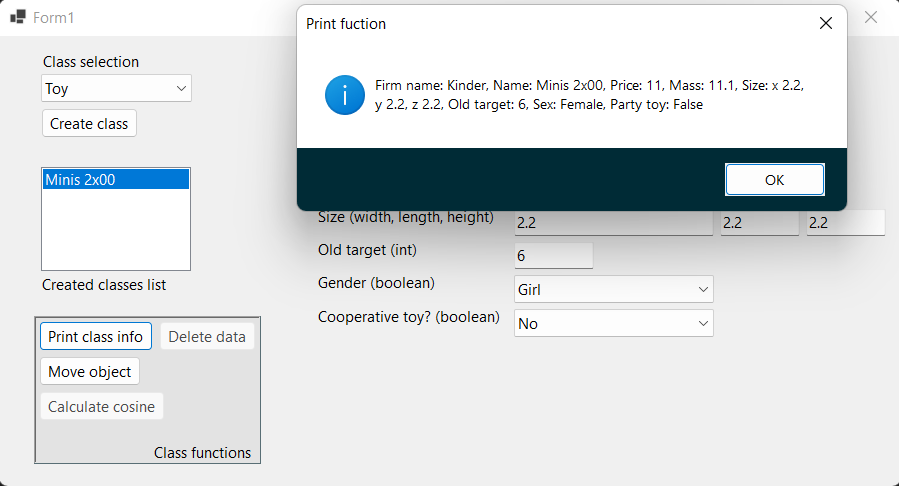
}

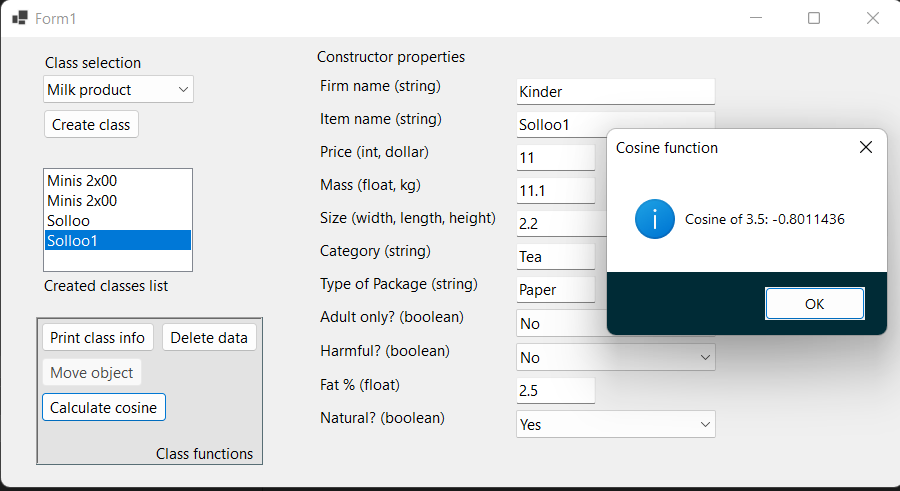
**UML-діаграма класів проекту**



**Демонстрація роботи програми**







## Практична робота № 9. «Події та делегати»

**Мета роботи:** засвоїти основи створення класів в мові C#, набути

практичних навичок застосування принципів поліморфізму об’єктно-

орієнтованого програмування.

Варіант 9

Реалізувати обробку помилок для практичної роботи №8, при цьому перевизначивши з допомогою наслідування події:

1. StackOverflowException
2. ArrayTypeMismatchException
3. DivideByZeroException
4. IndexOutOfRangeException
5. InvalidCastException
6. OutOfMemoryException
7. OverflowException

Делегат обробки виключень:

delegate void handle\_exception(string? message);

Події зловлення виключення:

event handle\_exception IndexOutOfRange\_e;

event handle\_exception ArrayTypeMismatch\_e;

event handle\_exception InvalidCast\_e;

event handle\_exception DivideByZero\_e;

event handle\_exception Overflow\_e;

Методи, які підписуються на події:

handle\_exception WrongIndex = (m) =>

{

MessageBox.Show($"You choose index out of range: {m}", "Exception handling {IndexOutOfRange}", MessageBoxButtons.OK, MessageBoxIcon.Error);

};

handle\_exception WrongTypeForArray= (m) =>

{

MessageBox.Show($"Attempt to add wrong type to array: {m}", "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

};

handle\_exception WrongCast = (m) =>

{

MessageBox.Show($"Cannot cast between your input to: {m}", "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

};

handle\_exception WrongDivideByZero = (m) =>

{

MessageBox.Show($"Error, you can't divide by zero", "Exception handling {DivideByZero}", MessageBoxButtons.OK, MessageBoxIcon.Error);

};

handle\_exception Overflow = (m) =>

{

MessageBox.Show($"Cannot add more max element handle: {m}", "Input Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

};

Лістинг підписання функцій на події:

public Form1()

{

InitializeComponent();

choose\_class\_combo.SelectedIndex = 0;

list = new List<item>();

IndexOutOfRange\_e += WrongIndex;

ArrayTypeMismatch\_e += WrongTypeForArray;

InvalidCast\_e += WrongCast;

DivideByZero\_e += WrongDivideByZero;

Overflow\_e += Overflow;

}

Лістинг зловлення виключень у коді:

IndexOutOfRangeException

try

{

if (class\_list.SelectedIndex == -1)

throw new IndexOutOfRangeException("[-1]");

Type t = list[class\_list.SelectedIndex].GetType();

if(typeof(product) == t || typeof(milk\_product) == t)

{

ISay\_butt.Enabled = true;

IErase\_butt.Enabled = true;

IMove\_butt.Enabled = false;

IMath\_butt.Enabled = false;

if (typeof(milk\_product) == t)

IMath\_butt.Enabled = true;

}

if (typeof(Toy) == t)

{

ISay\_butt.Enabled = true;

IMove\_butt.Enabled = true;

IMath\_butt.Enabled = false;

IErase\_butt.Enabled = false;

}

}catch(IndexOutOfRangeException exc) { IndexOutOfRange\_e(exc.Message); }

Якщо в елементі вікна class\_list обраний індекс -1, то викидається відповідне виключення

ArrayTypeMismatchException

try

{

switch (choose\_class\_combo.SelectedIndex)

{

case 0:

get\_product();

if (!created)

return;

list.Add(new product(cost, mass, size, name, firm\_name,

category, package, age\_b, health\_b));

break;

case 1:

get\_milk\_product();

if (!created)

return;

list.Add(new milk\_product(cost, mass, size, name, firm\_name,

category, package, age\_b, health\_b, fat, natural));

break;

case 2:

get\_toy();

if (!created)

return;

list.Add(new Toy(cost, mass, size, name, firm\_name,

old, boy\_b, party\_b));

break;

}

} catch(ArrayTypeMismatchException exc) { ArrayTypeMismatch\_e(exc.Message); }

Якщо в змінну list додати елемет, який не є потомком класу item, то викидається відповідне виключення.

InvalidCastException

try

{

if (!Int32.TryParse(constructor\_class\_table.Controls["Cost\_text"].Text, out int cost1))

{

MessageBox.Show("Wrong price input", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

created = false;

throw new InvalidCastException("int");

}

else

cost = cost1;

}catch(InvalidCastException exc) { InvalidCast\_e(exc.Message); }

Якщо у поле Cost введене некоректне значення, то викидається відповідне виключення.

DivideByZeroException

try

{

((milk\_product)list[class\_list.SelectedIndex]).GetCosine();

} catch(DivideByZeroException exc) { DivideByZero\_e(exc.Message); }

Клас class milk\_product : product, IMath реалізує інтерфейс IMath таким чином:

public float GetCosine()

{

if (Convert.ToInt32(Natural) == 0)

throw new DivideByZeroException();

float res = (float)Math.Cos(FatPercent/Convert.ToInt32(Natural));

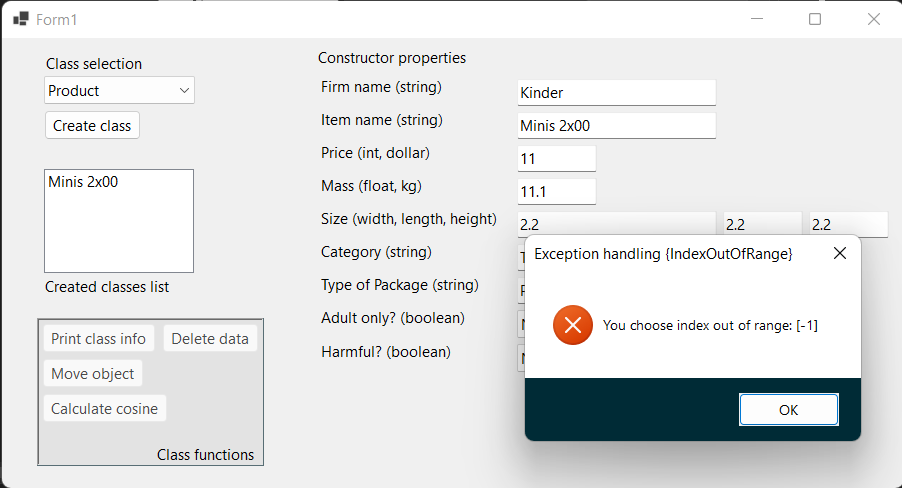
MessageBox.Show($"Cosine of {FatPercent + Convert.ToInt32(Natural)}: {res}", "Cosine function", MessageBoxButtons.OK, MessageBoxIcon.Information);

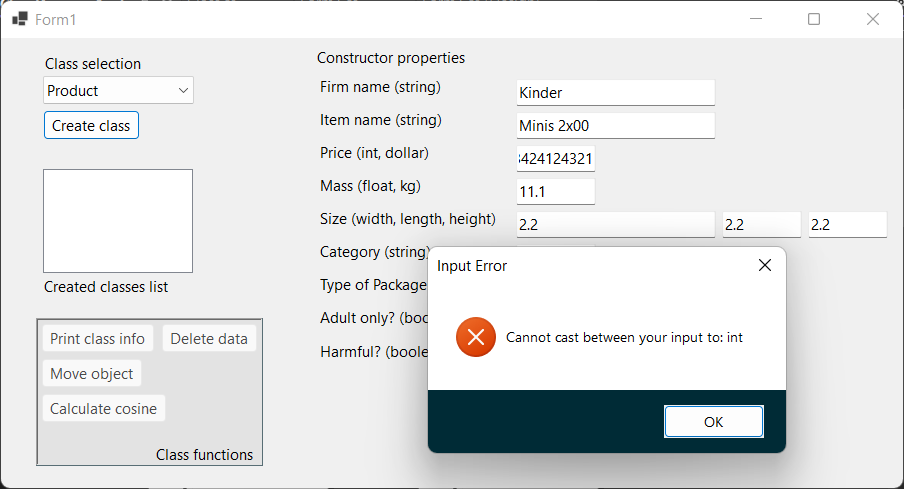
return res;

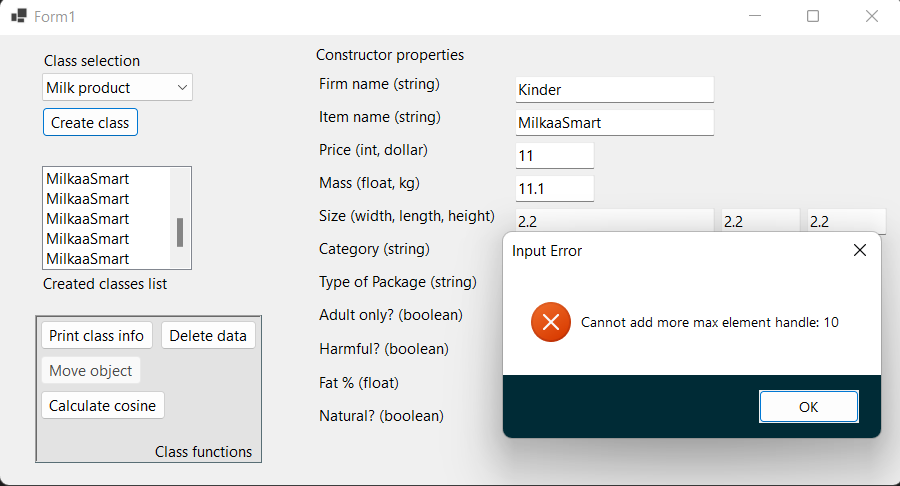
}

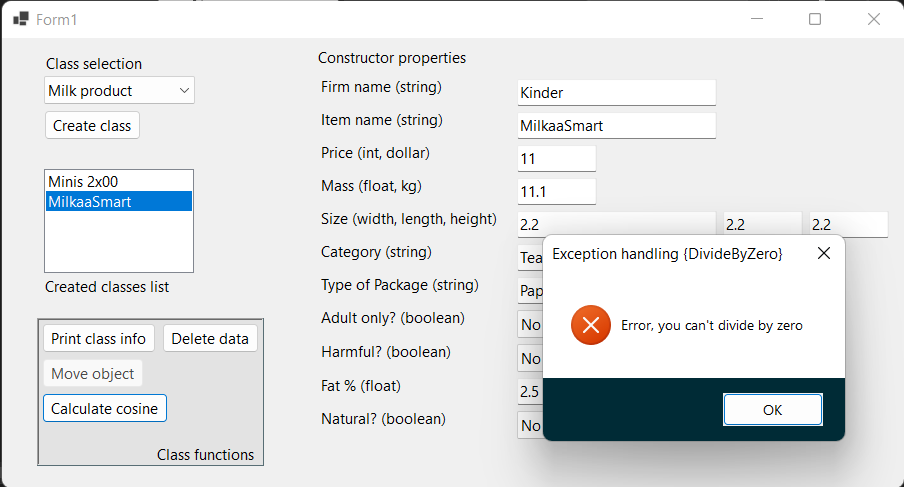
І якщо змінна Natural дорівнює false, то при рахуванні синусу число буле ділитися на 0. Щоб цьому запобігти викидається виключення DivideByZeroException.

**Приклад оброблення та повідомлення користувачу про виключення**









# ВИСНОВКИ

В ході опрацювання практичних робіт було виконано ряд важливих пунктів:

* Набути навички програмування мовою C#;
* Закріпити вміння створювати і реалізовувати ієрархію класів в мові C#, набути практичні навички застосування принципів поліморфізму об’єктно-орієнтованого програмування.
* Засвоїти основи використання масивів та колекцій, закріпити навички роботи з двовимірними масивами.
* Навчитися створювати інтерфейси та віртуальні класи при розробці проєктів мовою С#.
* Набути навички використання делегатів та подій та навчитися застосовувати різні шаблони функцій для вирішення задач.

Підсумовуючи вище сказане, можна впевнено стверджувати, що під час навчальної практики були закріплені знання з об'єктно-орієнтованого програмування та основ програмної інженерії.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Страуструп Б. Язык программирования С#. Специальное издание.-М., СПб.: «Издательство БИНОМ» – «Невский Диалект», 2001г.-1099 с.

2. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на С#. Пер. с англ. – М.: СПб.: «Издательство БИНОМ»-«Невский Диалект», 2001.-560 с.

3. Глушаков С.В., Коваль А.В., Смирнов С.В. Язык программирования С#: Учебный курс.- Харьков: Фолио; М.: «Видавництво АСТ», 2001.-500 с.

4. Р. Лафоре. Объектно-ориентированное программирование в С#. 4-е издание. Издательство: Питер. Серия: Классика computer science, 2005.- 928 с.

5. Шилдт Г. Самоучитель С#, 3-е издание: пер. с англ.-СПб.:ВНV-Санкт-Петербург,1999.-688 с.

# ЩОДЕННИК ПРАКТИКАНТА

Навчальної практики з об’єктно-орієнтованого програмування

**Студента(ки)** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Групи \_\_\_\_**РПЗ-93Б\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Спеціальності \_\_**121 – « Інженерія програмного забезпечення»\_\_\_\_\_\_\_\_

**Кваліфікаційний рівень** \_\_\_молодший бакалавр\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**База практики**  Київський фаховий коледж зв’язку\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| № | Дата | Зміст роботи | Підпис керівника |
| 1 | 30.05.22 | Ознайомлення з технікою безпеки і задачами практики |  |
| 2 | 31.05.22 –02.06.22 | Розробка алгоритму і написання коду програм до практичної роботи № 1 |  |
| 3 | 03.06.22 | Опрацювання літератури |  |
| 3 | 06.06.22 –08.06.22 | Розробка алгоритму і написання коду програми до практичної роботи № 2 |  |
| 4 | 09.06.22–10.06.22 | Розробка алгоритму і написання коду для практичної роботи № 3 |  |
| 5 | 13.06.22–15.06.22 | Розробка алгоритму і написання коду програми до практичної роботи № 4 |  |
| 6 | 16.06.22–18.06.22 | Розробка алгоритму і написання коду програми до практичної роботи № 5 |  |
| 7 | 20.06.22 – 21.06.22 | Розробка алгоритму і написання коду програми до практичної роботи № 6 |  |
| 8 | 22.06.22 – 23.06.22 | Розробка алгоритму і написання коду програми до практичної роботи № 8 |  |
| 9 | 24.06.22 | Розробка алгоритму і написання коду програми до практичної роботи № 9 |  |
| 10 | 13.06.22–24.06.22 | Оформлення звіту |  |
| 11 | 24.06.22 | Захист звіту |  |

Керівник практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ З. О. Ольшевська

**КИЇВСЬКИЙ КОЛЕДЖ ЗВ’ЯЗКУ**

Циклова комісія «Програмної інженерії та прикладної математики»

# Критерії обліку знань студентів

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Шкала в балах | 90-100 | 82-89 | 75-81 | 67-74 | 60-66 | 35-59 | 1-34 |
| Шкала ECTS  (оцінка) | A | B | C | D | T | FX | F |
| Нац.шкала  (оцінка) | відмінно | Добре | | задовільно | | незадовільно | |

**Оцінювання знань студента(ки)** \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_ \_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Вид діяльності | Коефіцієнт вартості(бали) | Кількість робіт | Результат (бали) |
| 1 | Відвідування занять | 1 | 25 |  |
| 2 | Виконання завдань | 5 | 10 |  |
| 3 | Звіт | 10 | 1 |  |
| 4 | Захист | 15 | 1 |  |
| Всього: | | | |  |

**Примітка**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**