**КИЇВСЬКИЙ ФАХОВИЙ КОЛЕДЖ ЗВ’ЯЗКУ**

Циклова комісія «Програмної інженерії та прикладної математики»

**ЗВІТ**

**ПРО ВИКОНАННЯ НАВЧАЛЬНОЇ ПРАКТИКИ З ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ**

**Студента**\_\_Філіпенко Андрій Дмитрович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(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)

**[ПРАКТИЧНА РОБОТА №1 4](#ПР1)**

**[ПРАКТИЧНА РОБОТА №2 33](#ПР2)**

**[ПРАКТИЧНА РОБОТА №3 42](#ПР3)**

**[ПРАКТИЧНА РОБОТА №4 54](#ПР4)**

**[ПРАКТИЧНА РОБОТА №5 61](#ПР5)**

**[ПРАКТИЧНА РОБОТА №6 76](#ПР6)**

**[ПРАКТИЧНА РОБОТА №7 79](#ПР7)**

**[ПРАКТИЧНА РОБОТА №8 84](#ПР8)**

**[ПРАКТИЧНА РОБОТА №9 88](#ПР9)**

[**ПЕРЕЛІК ПОСИЛАНЬ**](#посил) **93**

**[ЩОДЕННИК ПРАКТИКАНТА 94](#щод)**

**[КРИТЕРІЇ ОБЛІКУ ЗНАНЬ СТУДЕНТІВ 95](#критерії)**

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.

1. Опис класів проекту:
   1. Клас *Frаction*:

Повний лістинг класу:

public class Fraction : IComparable

{

private int \_sign;

private int \_intPart;

private int \_numerator;

private int \_denominator;

public int Numerator { set { \_numerator = value; } get { return \_numerator; } }

public int Denominator { set { \_denominator = value; } get { return \_denominator; } }

public int IntPart { set { \_intPart = value; } get => GetIntPart(); }

public int Sign { set { \_sign = value; } get { return \_sign; } }

public Fraction()

{

}

public Fraction(int numerator, int denominator, int sign = 1, int intPart = 0)

{

\_sign = sign;

\_intPart = intPart;

\_numerator = numerator;

\_denominator = denominator;

}

public Fraction(double number)

{

if (number >= 0)

{

\_sign = 1;

}

else \_sign = -1;

string str = number.ToString();

string[] str\_array = str.Split(",");

\_intPart = Math.Abs(Convert.ToInt32(str\_array[0]));

\_numerator = Convert.ToInt32(str\_array[1]);

string zeros = "";

for (int i = 0; i < (str\_array[1].Length); i++)

{

zeros += "0";

}

\_denominator = Convert.ToInt32("1" + zeros);

}

public void GetImproperView()

{

\_numerator += \_intPart \* \_denominator;

\_intPart = 0;

}

public void GetMixedView()

{

Cancellation();

if (\_numerator > \_denominator)

{

\_intPart = \_numerator / \_denominator;

\_numerator = \_numerator - (\_denominator \* \_intPart);

}

}

public void Cancellation()

{

bool access = false;

while (!access)

{

if (\_numerator % 2 == 0 && \_denominator % 2 == 0)

{

\_numerator = \_numerator / 2;

\_denominator = \_denominator / 2;

}

else if (\_numerator % 3 == 0 && \_denominator % 3 == 0)

{

\_numerator = \_numerator / 3;

\_denominator = \_denominator / 3;

}

else if (\_numerator % 5 == 0 && \_denominator % 5 == 0)

{

\_numerator = \_numerator / 5;

\_denominator = \_denominator / 5;

}

else access = true;

}

}

private int GetIntPart()

{

GetMixedView();

return \_intPart;

}

public static Fraction Parse(string str)

{

int intPart, numerator, denominator, sign;

string[] strs = str.Split(' ');

string[] strs1;

Fraction res;

if (strs.Length == 1)

{

strs1 = strs[0].Split('/');

numerator = int.Parse(strs1[0]);

denominator = int.Parse(strs1[1]);

res = new Fraction(numerator, denominator);

res.GetMixedView();

return res;

}

sign = int.Parse(strs[0]);

intPart = int.Parse(strs[1]);

numerator = int.Parse(strs[2]);

denominator = int.Parse(strs[3]);

res = new Fraction(numerator, denominator, intPart, sign);

res.GetMixedView();

return res;

}

public static Fraction operator +(Fraction fraction1, Fraction fraction2)

{

Fraction result = new Fraction();

result.IntPart = fraction1.Sign \* fraction1.IntPart + fraction2.Sign \* fraction2.IntPart;

result.Numerator = fraction1.Sign \* (fraction1.Numerator \* fraction2.Denominator)

+ fraction2.Sign \* (fraction2.Numerator \* fraction1.Denominator);

result.Denominator = fraction1.Denominator \* fraction2.Denominator;

result.Cancellation();

return result;

}

public static Fraction operator +(Fraction fraction, int number)

{

fraction.IntPart += number;

return fraction;

}

public static float operator +(float number, Fraction fraction)

{

number += fraction.IntPart;

number += ((float)fraction.Numerator / (float)fraction.Denominator);

return number;

}

public static Fraction operator -(Fraction fraction)

{

fraction.Sign \*= -1;

return fraction;

}

public static Fraction operator -(Fraction fraction1, Fraction fraction2)

{

Fraction result = new Fraction();

result.IntPart = fraction1.Sign \* fraction1.IntPart - fraction2.Sign \* fraction2.IntPart;

result.Numerator = fraction1.Sign \* (fraction1.Numerator \* fraction2.Denominator)

- fraction2.Sign \* (fraction2.Numerator \* fraction1.Denominator);

result.Denominator = fraction1.Denominator \* fraction2.Denominator;

result.Cancellation();

return result;

}

public static Fraction operator -(Fraction fraction, int number)

{

fraction.IntPart -= number;

return fraction;

}

public static float operator -(float number, Fraction fraction)

{

number -= fraction.IntPart;

number -= ((float)fraction.Numerator / (float)fraction.Denominator);

return number;

}

public static Fraction operator \*(Fraction fraction1, Fraction fraction2)

{

Fraction result = new Fraction();

fraction1.GetImproperView();

fraction2.GetImproperView();

result.Numerator = fraction1.Numerator \* fraction2.Numerator;

result.Denominator = fraction1.Denominator \* fraction2.Denominator;

result.GetMixedView();

return result;

}

public static Fraction operator \*(Fraction fraction, int number)

{

fraction.GetImproperView();

fraction.Numerator \*= number;

fraction.GetMixedView();

return fraction;

}

public static float operator \*(float number, Fraction fraction)

{

fraction.GetImproperView();

number = (number \* fraction.Numerator) / (float)fraction.Denominator;

return number;

}

public static Fraction operator /(Fraction fraction1, Fraction fraction2)

{

Fraction result = new Fraction();

fraction1.GetImproperView();

fraction2.GetImproperView();

result.Numerator = fraction1.Numerator \* fraction2.Denominator;

result.Denominator = fraction1.Denominator \* fraction2.Numerator;

result.GetMixedView();

return result;

}

public static Fraction operator /(Fraction fraction, int number)

{

fraction.Denominator \*= number;

fraction.GetMixedView();

return fraction;

}

public static float operator /(float number, Fraction fraction)

{

fraction.GetImproperView();

number = (number \* fraction.Denominator) / (float)fraction.Numerator;

return number;

}

public static bool operator >(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator > (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

public static bool operator <(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator < (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

public static bool operator >=(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator >= (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

public static bool operator <=(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator <= (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

public static bool operator !=(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator != (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

public static bool operator ==(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator == (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

public int CompareTo(object obj)

{

if (this < (obj as Fraction)) return -1;

if (this > (obj as Fraction)) return 1;

return 0;

}

public override string ToString()

{

string sgn = "";

if (\_sign == -1) sgn = "-";

return "Integer part: "+ sgn + IntPart + " Fraction: " + Numerator + "/" + Denominator;

}

~Fraction()

{

Console.WriteLine("Object has destroyed");

}

}

}

На початку створюється декілька приватних змінних для роботи з дробами та проводиться їх інкапсуляція.

private int \_sign;

private int \_intPart;

private int \_numerator;

private int \_denominator;

public int Numerator { set { \_numerator = value; } get { return \_numerator; } }

public int Denominator { set { \_denominator = value; } get { return \_denominator; } }

public int IntPart { set { \_intPart = value; } get => GetIntPart(); }

public int Sign { set { \_sign = value; } get { return \_sign; } }

Описується 3 конструктори класу Fraction, перший пустий, в другий передаються змінні int numerator, int denominator, int sign = 1, int intPart=0

, в третій double number де відбувається перетворення дійсного числа до типу *Fraction*..

public Fraction()

{

}

public Fraction(int numerator, int denominator, int sign = 1, int intPart = 0)

{

\_sign = sign;

\_intPart = intPart;

\_numerator = numerator;

\_denominator = denominator;

}

public Fraction(double number)

{

if (number >= 0)

{

\_sign = 1;

}

else \_sign = -1;

string str = number.ToString();

string[] str\_array = str.Split(",");

\_intPart = Math.Abs(Convert.ToInt32(str\_array[0]));

\_numerator = Convert.ToInt32(str\_array[1]);

string zeros = "";

for (int i = 0; i < (str\_array[1].Length); i++)

{

zeros += "0";

}

\_denominator = Convert.ToInt32("1" + zeros);

}

Метод перетворення дробу у неправильний:

public void GetImproperView()

{

\_numerator += \_intPart \* \_denominator;

\_intPart = 0;

}

Метод перетворення дробу у змішаний(виділення цілої частини):

public void GetMixedView()

{

Cancellation();

if (\_numerator > \_denominator)

{

\_intPart = \_numerator / \_denominator;

\_numerator = \_numerator - (\_denominator \* \_intPart);

}

}

Метод спрощення дробу. За трьома основними параметрами(ділення на 2, 3 та 5 без остачі) дріб спрощується:

public void Cancellation()

{

bool access = false;

while (!access)

{

if (\_numerator % 2 == 0 && \_denominator % 2 == 0)

{

\_numerator = \_numerator / 2;

\_denominator = \_denominator / 2;

}

else if (\_numerator % 3 == 0 && \_denominator % 3 == 0)

{

\_numerator = \_numerator / 3;

\_denominator = \_denominator / 3;

}

else if (\_numerator % 5 == 0 && \_denominator % 5 == 0)

{

\_numerator = \_numerator / 5;

\_denominator = \_denominator / 5;

}

else access = true;

}

}

Метод, який виділяє цілу частину з дробу:

private int GetIntPart()

{

GetMixedView();

return \_intPart;

}

Статичний метод *Parse* класу *Fraction*, що розділяє строку на екземпляр класу:

public static Fraction Parse(string str)

{

int intPart, numerator, denominator, sign;

string[] strs = str.Split(' ');

string[] strs1;

Fraction res;

if (strs.Length == 1)

{

strs1 = strs[0].Split('/');

numerator = int.Parse(strs1[0]);

denominator = int.Parse(strs1[1]);

res = new Fraction(numerator, denominator);

res.GetMixedView();

return res;

}

sign = int.Parse(strs[0]);

intPart = int.Parse(strs[1]);

numerator = int.Parse(strs[2]);

denominator = int.Parse(strs[3]);

res = new Fraction(numerator, denominator, intPart, sign);

res.GetMixedView();

return res;

}

Перегрузки арифметичних операторів та операторів порівняння для дій між двома різними екземплярами класу *Fraction*.

public static Fraction operator +(Fraction fraction1, Fraction fraction2)

{

Fraction result = new Fraction();

result.IntPart = fraction1.Sign \* fraction1.IntPart + fraction2.Sign \* fraction2.IntPart;

result.Numerator = fraction1.Sign \* (fraction1.Numerator \* fraction2.Denominator)

+ fraction2.Sign \* (fraction2.Numerator \* fraction1.Denominator);

result.Denominator = fraction1.Denominator \* fraction2.Denominator;

result.Cancellation();

return result;

}

public static Fraction operator +(Fraction fraction, int number)

{

fraction.IntPart += number;

return fraction;

}

public static float operator +(float number, Fraction fraction)

{

number += fraction.IntPart;

number += ((float)fraction.Numerator / (float)fraction.Denominator);

return number;

}

public static Fraction operator -(Fraction fraction)

{

fraction.Sign \*= -1;

return fraction;

}

public static Fraction operator -(Fraction fraction1, Fraction fraction2)

{

Fraction result = new Fraction();

result.IntPart = fraction1.Sign \* fraction1.IntPart - fraction2.Sign \* fraction2.IntPart;

result.Numerator = fraction1.Sign \* (fraction1.Numerator \* fraction2.Denominator)

- fraction2.Sign \* (fraction2.Numerator \* fraction1.Denominator);

result.Denominator = fraction1.Denominator \* fraction2.Denominator;

result.Cancellation();

return result;

}

public static Fraction operator -(Fraction fraction, int number)

{

fraction.IntPart -= number;

return fraction;

}

public static float operator -(float number, Fraction fraction)

{

number -= fraction.IntPart;

number -= ((float)fraction.Numerator / (float)fraction.Denominator);

return number;

}

public static Fraction operator \*(Fraction fraction1, Fraction fraction2)

{

Fraction result = new Fraction();

fraction1.GetImproperView();

fraction2.GetImproperView();

result.Numerator = fraction1.Numerator \* fraction2.Numerator;

result.Denominator = fraction1.Denominator \* fraction2.Denominator;

result.GetMixedView();

return result;

}

public static Fraction operator \*(Fraction fraction, int number)

{

fraction.GetImproperView();

fraction.Numerator \*= number;

fraction.GetMixedView();

return fraction;

}

public static float operator \*(float number, Fraction fraction)

{

fraction.GetImproperView();

number = (number \* fraction.Numerator) / (float)fraction.Denominator;

return number;

}

public static Fraction operator /(Fraction fraction1, Fraction fraction2)

{

Fraction result = new Fraction();

fraction1.GetImproperView();

fraction2.GetImproperView();

result.Numerator = fraction1.Numerator \* fraction2.Denominator;

result.Denominator = fraction1.Denominator \* fraction2.Numerator;

result.GetMixedView();

return result;

}

public static Fraction operator /(Fraction fraction, int number)

{

fraction.Denominator \*= number;

fraction.GetMixedView();

return fraction;

}

public static float operator /(float number, Fraction fraction)

{

fraction.GetImproperView();

number = (number \* fraction.Denominator) / (float)fraction.Numerator;

return number;

}

public static bool operator >(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator > (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

public static bool operator <(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator < (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

public static bool operator >=(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator >= (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

public static bool operator <=(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator <= (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

public static bool operator !=(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator != (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

public static bool operator ==(Fraction ob1, Fraction ob2)

{

bool result;

ob1.GetImproperView();

ob2.GetImproperView();

result = (float)ob1.Numerator / (float)ob1.Denominator == (float)ob2.Numerator / (float)ob2.Denominator;

ob1.GetMixedView();

ob2.GetMixedView();

return result;

}

Інтерфейс для сортування класу *Fraction*:

public int CompareTo(object obj)

{

if (this < (obj as Fraction)) return -1;

if (this > (obj as Fraction)) return 1;

return 0;

}

Перевизначення методу *ToString* базового класу *Object:*

public override string ToString()

{

string sgn = "";

if (\_sign == -1) sgn = "-";

return "Integer part: "+ sgn + IntPart + " Fraction: " + Numerator + "/" + Denominator;

}

Деструктор класу *Fraction*

~Fraction()

{

Console.WriteLine("Object has destroyed");

}

* 1. Клас *Complex*:

Повний лістинг класу:

internal class Complex

{

private int \_a = 0;

private int \_b = 0;

public int Re { set => \_a = value; get => \_a; }

public int Im { set => \_b = value; get => \_b; }

public Complex()

{

}

public Complex(int a, int b)

{

\_a = a;

\_b = b;

}

~Complex()

{

Console.WriteLine("Object has destroyed.");

}

public static Complex operator +(Complex u, Complex v)

{

u.\_a += v.\_a;

u.\_b += v.\_b;

return u;

}

public static Complex operator -(Complex u, Complex v)

{

u.\_a -= v.\_a;

u.\_b -= v.\_b;

return u;

}

public static Complex operator \*(Complex u, Complex v)

{

u.\_a = (u.\_a \* v.\_a) - (u.\_b \* v.\_b);

v.\_b = (u.\_b \* v.\_a) + (u.\_a \* v.\_b);

return u;

}

public static Complex operator /(Complex u, Complex v)

{

u.\_a = ((u.\_a \* v.\_a) + (u.\_b \* v.\_b)) / (v.\_a^2 + v.\_b^2);

v.\_b = ((u.\_b\*v.\_a)-(u.\_a\*v.\_b)) / (v.\_a ^ 2 + v.\_b ^ 2);

return u;

}

public static Complex Parse(string str)

{

Complex res = new Complex();

string[] str\_array = str.Split(' ');

res.\_a = Convert.ToInt32(str\_array[0]);

res.\_b = Convert.ToInt32(str\_array[2].Trim('i'));

return res;

}

public override string ToString()

{

if (\_b > 0) return \_a + "+" + \_b + "i";

else if (\_b == 0) return Convert.ToString(\_a);

else if (\_a == 0) return \_b + "i";

else if (\_b < 0) return \_a +""+ \_b + "i";

else return "Error";

}

}

Задання змінних та їх інкапсуляція:

private int \_a = 0;

private int \_b = 0;

public int Re { set => \_a = value; get => \_a; }

public int Im { set => \_b = value; get => \_b; }

Конструктори та дуструктор класу *Complex:*

public Complex()

{

}

public Complex(int a, int b)

{

\_a = a;

\_b = b;

}

~Complex()

{

Console.WriteLine("Object has destroyed.");

}

Перегрузки арифметичних операторів для дій між екземплярами класу Complex:

public static Complex operator +(Complex u, Complex v)

{

u.\_a += v.\_a;

u.\_b += v.\_b;

return u;

}

public static Complex operator -(Complex u, Complex v)

{

u.\_a -= v.\_a;

u.\_b -= v.\_b;

return u;

}

public static Complex operator \*(Complex u, Complex v)

{

u.\_a = (u.\_a \* v.\_a) - (u.\_b \* v.\_b);

v.\_b = (u.\_b \* v.\_a) + (u.\_a \* v.\_b);

return u;

}

public static Complex operator /(Complex u, Complex v)

{

u.\_a = ((u.\_a \* v.\_a) + (u.\_b \* v.\_b)) / (v.\_a^2 + v.\_b^2);

v.\_b = ((u.\_b\*v.\_a)-(u.\_a\*v.\_b)) / (v.\_a ^ 2 + v.\_b ^ 2);

return u;

}

Статичний метод *Parse* класу *Complex*, що розділяє строку на екземпляр класу:

public static Complex Parse(string str)

{

Complex res = new Complex();

string[] str\_array = str.Split(' ');

res.\_a = Convert.ToInt32(str\_array[0]);

res.\_b = Convert.ToInt32(str\_array[2].Trim('i'));

return res;

}

Перевизначення методу *ToString* базового класу *Object:*

public override string ToString()

{

if (\_b > 0) return \_a + "+" + \_b + "i";

else if (\_b == 0) return Convert.ToString(\_a);

else if (\_a == 0) return \_b + "i";

else if (\_b < 0) return \_a +""+ \_b + "i";

else return "Error";

}

* 1. Клас *Date*:

Повний лістинг класу:

internal class Date

{

public int day;

public int month;

public int year;

private int maxdays;

private static int[] maxdays30 = { 4, 6, 9, 11 };

private static int[] maxdays31 = { 1, 3, 5, 7, 8, 10, 12 };

public bool isLeap { private set; get; }

public Date()

{

}

public Date(int \_day, int \_month, int \_year)

{

year = \_year;

if (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0))

isLeap = true;

else

isLeap = false;

while (\_month > 12)

{

\_month -= 12;

year++;

}

if (\_month < 1)

\_month = 1;

month = \_month;

day = checkDays(\_day);

}

private int checkDays(int day)

{

if (maxdays31.Contains(month))

{

maxdays = 31;

day = checkMaxDays(day);

}

else if (maxdays30.Contains(month))

{

maxdays = 30;

day = checkMaxDays(day);

}

else if (month == 2 && isLeap)

{

maxdays = 29;

day = checkMaxDays(day);

}

else if (month == 2 && !isLeap)

{

maxdays = 28;

day = checkMaxDays(day);

}

return day;

}

private int checkMaxDays(int day)

{

if (day > maxdays)

{

day -= maxdays;

month++;

while (month > 12)

{

month -= 12;

year++;

}

day = checkDays(day);

}

return day;

}

public static Date Parse(string str)

{

string[] array = str.Split(".");

int parsedDay = Convert.ToInt32(array[0]);

int parsedMonth = Convert.ToInt32(array[1]);

int parsedYear = Convert.ToInt32(array[2]);

Date date = new Date(parsedDay,parsedMonth,parsedYear);

return date;

}

public static Date operator +(Date date1, Date date2)

{

Date result = new Date();

result.year = date1.year + date2.year;

result.month = date1.month + date2.month;

while (result.month > 12)

{

result.month-=12;

result.year++;

}

result.day = date1.day + date2.day;

result.day = result.checkDays(result.day);

return result;

}

public static Date operator +(Date date, int days)

{

date.day += days;

date.day = date.checkDays(date.day);

return date;

}

public static bool operator >(Date date1, Date date2)

{

if (date1.year > date2.year)

return true;

else if (date1.month > date2.month)

return true;

else if (date1.day>date2.day)

return true;

else

return false;

}

public static bool operator <(Date date1, Date date2)

{

if (date1.year < date2.year)

return true;

else if (date1.month < date2.month)

return true;

else if (date1.day < date2.day)

return true;

else

return false;

}

public static bool operator >=(Date date1, Date date2)

{

if (date1.year >= date2.year)

return true;

else if (date1.month >= date2.month)

return true;

else if (date1.day >= date2.day)

return true;

else

return false;

}

public static bool operator <=(Date date1, Date date2)

{

if (date1.year <= date2.year)

return true;

else if (date1.month <= date2.month)

return true;

else if (date1.day <= date2.day)

return true;

else

return false;

}

public static bool operator ==(Date date1, Date date2)

{

if (date1.year == date2.year && date1.month == date2.month && date1.day == date2.day)

return true;

else

return false;

}

public static bool operator !=(Date date1, Date date2)

{

if (date1.year != date2.year && date1.month != date2.month && date1.day != date2.day)

return true;

else

return false;

}

public override string ToString()

{

return "Day: " + day + " Month: " + month + " Year: " + year;

}

public override bool Equals(object obj)

{

return obj is Date date &&

day == date.day &&

month == date.month &&

year == date.year;

}

public override int GetHashCode()

{

return HashCode.Combine(day, month, year);

}

}

Створення змінних та задання властивості одній з них:

public int day;

public int month;

public int year;

private int maxdays;

private static int[] maxdays30 = { 4, 6, 9, 11 };

private static int[] maxdays31 = { 1, 3, 5, 7, 8, 10, 12 };

public bool isLeap { private set; get; }

Конструтори класу *Date,* у другому відбувається перевірка на високосний рік та скорочення отриманих даних до нормального формату:

public Date()

{

}

public Date(int \_day, int \_month, int \_year)

{

year = \_year;

if (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0))

isLeap = true;

else

isLeap = false;

while (\_month > 12)

{

\_month -= 12;

year++;

}

if (\_month < 1)

\_month = 1;

month = \_month;

day = checkDays(\_day);

}

Метод *CheckDays* за заданими умовами викликає метод *checkMaxDays* для перевірки, чи може поточний місяць містити дану кількість днів:

private int checkDays(int day)

{

if (maxdays31.Contains(month))

{

maxdays = 31;

day = checkMaxDays(day);

}

else if (maxdays30.Contains(month))

{

maxdays = 30;

day = checkMaxDays(day);

}

else if (month == 2 && isLeap)

{

maxdays = 29;

day = checkMaxDays(day);

}

else if (month == 2 && !isLeap)

{

maxdays = 28;

day = checkMaxDays(day);

}

return day;

}

Метод *checkMaxDays* виконує порівяння максимольно можливої к-сті днів, заданої у методі *CheckDays* та поточної кількості днів. Якщо поточних днів більше можливого, відбувається рекурсія методу *CheckDays*:

private int checkMaxDays(int day)

{

if (day > maxdays)

{

day -= maxdays;

month++;

while (month > 12)

{

month -= 12;

year++;

}

day = checkDays(day);

}

return day;

}

Статичний метод *Parse* класу *Date*, що розділяє строку на екземпляр класу:

public static Date Parse(string str)

{

string[] array = str.Split(".");

int parsedDay = Convert.ToInt32(array[0]);

int parsedMonth = Convert.ToInt32(array[1]);

int parsedYear = Convert.ToInt32(array[2]);

Date date = new Date(parsedDay,parsedMonth,parsedYear);

return date;

}

Перегрузки арифметичних операторів та операторів порівняння для дій між двома різними екземплярами класу *Date*:

public static Date operator +(Date date1, Date date2)

{

Date result = new Date();

result.year = date1.year + date2.year;

result.month = date1.month + date2.month;

while (result.month > 12)

{

result.month-=12;

result.year++;

}

result.day = date1.day + date2.day;

result.day = result.checkDays(result.day);

return result;

}

public static Date operator +(Date date, int days)

{

date.day += days;

date.day = date.checkDays(date.day);

return date;

}

public static bool operator >(Date date1, Date date2)

{

if (date1.year > date2.year)

return true;

else if (date1.month > date2.month)

return true;

else if (date1.day>date2.day)

return true;

else

return false;

}

public static bool operator <(Date date1, Date date2)

{

if (date1.year < date2.year)

return true;

else if (date1.month < date2.month)

return true;

else if (date1.day < date2.day)

return true;

else

return false;

}

public static bool operator >=(Date date1, Date date2)

{

if (date1.year >= date2.year)

return true;

else if (date1.month >= date2.month)

return true;

else if (date1.day >= date2.day)

return true;

else

return false;

}

public static bool operator <=(Date date1, Date date2)

{

if (date1.year <= date2.year)

return true;

else if (date1.month <= date2.month)

return true;

else if (date1.day <= date2.day)

return true;

else

return false;

}

public static bool operator ==(Date date1, Date date2)

{

if (date1.year == date2.year && date1.month == date2.month && date1.day == date2.day)

return true;

else

return false;

}

public static bool operator !=(Date date1, Date date2)

{

if (date1.year != date2.year && date1.month != date2.month && date1.day != date2.day)

return true;

else

return false;

}

Перевизначення методу *ToString* базового класу *Object:*

public override string ToString()

{

return "Day: " + day + " Month: " + month + " Year: " + year;

}

* 1. Клас *Time*:

Повний лістинг класу:

public class Time

{

private int \_seconds = 0;

private int \_minutes = 0;

private int \_hours = 0;

public int Seconds { set => \_seconds = value; get => \_seconds; }

public int Minutes { set => \_minutes = value; get => \_minutes; }

public int Hours { set => \_hours = value; get => \_hours; }

public Time()

{

}

public Time(int seconds, int minutes, int hours)

{

Check(seconds,minutes,hours);

}

public Time(int seconds)

{

\_hours = seconds / 60 / 60;

\_minutes = (seconds / 60) - (\_hours \*60);

\_seconds = seconds-((\_minutes \* 60)+(\_hours \* 60\*60));

}

public static Time operator + (Time time, int minutes)

{

time.\_minutes += minutes;

if(time.\_minutes>60)

{

time.\_hours++;

time.\_minutes -= 60;

}

return time;

}

public void Check(int seconds, int minutes, int hours)

{

if (seconds < 60)

{

\_seconds += seconds;

}

else

{

seconds -= 60;

\_seconds += seconds;

\_minutes++;

}

if (minutes < 60)

{

\_minutes += minutes;

}

else

{

minutes -= 60;

\_minutes += minutes;

\_hours++;

}

\_hours += hours;

}

public void Check()

{

if(\_seconds>60)

{

\_seconds -= 60;

\_minutes++;

}

if(\_minutes>60)

{

\_minutes -= 60;

\_hours++;

}

if(\_seconds<0)

{

\_seconds += 60;

\_minutes--;

}

if(\_minutes<0)

{

\_minutes += 60;

\_hours--;

}

if(\_hours<0)

{

Console.WriteLine("Error");

}

}

public static Time operator - (Time time\_start, Time time\_finish)

{

time\_start.\_seconds -= time\_finish.\_seconds;

time\_start.\_minutes -= time\_finish.\_minutes;

time\_start.\_hours -= time\_finish.\_hours;

time\_start.Check();

return time\_start;

}

public static Time Parse(string time)

{

string[] str = time.Split(' ');

Time t = new Time();

if(str.Length == 1)

{

int seconds = int.Parse(str[0]);

t.\_hours = seconds / 60 / 60;

t.\_minutes = (seconds / 60) - (t.\_hours \* 60);

t.\_seconds = seconds - ((t.\_minutes \* 60) + (t.\_hours \* 60 \* 60));

t.Check();

return t;

}

t.\_seconds = int.Parse(str[0]);

t.\_minutes = int.Parse(str[1]);

t.\_hours = int.Parse(str[2]);

t.Check();

return t;

}

public override string ToString()

{

return "Hours: "+\_hours+" Minutes: "+\_minutes+" Seconds: "+\_seconds;

}

~Time()

{

Console.WriteLine("Object has destroyed.");

}

}

Задання змінних та їх інкапсуляція:

private int \_seconds = 0;

private int \_minutes = 0;

private int \_hours = 0;

public int Seconds { set => \_seconds = value; get => \_seconds; }

public int Minutes { set => \_minutes = value; get => \_minutes; }

public int Hours { set => \_hours = value; get => \_hours; }

Три конструктори класу *Time* для різного формату вводу даних:

public Time()

{

}

public Time(int seconds, int minutes, int hours)

{

Check(seconds,minutes,hours);

}

public Time(int seconds)

{

\_hours = seconds / 60 / 60;

\_minutes = (seconds / 60) - (\_hours \*60);

\_seconds = seconds-((\_minutes \* 60)+(\_hours \* 60\*60));

}

Перегрузка оператора суми для додавання цілочисельного значення до екземпляру класу *Time*:

public static Time operator + (Time time, int minutes)

{

time.\_minutes += minutes;

if(time.\_minutes>60)

{

time.\_hours++;

time.\_minutes -= 60;

}

return time;

}

Оператор *Check*, який отримує три змінних(секунди, хвилини години) та проводить приведення їх до нормального формату. Він використувується у дрігому констркторі класу *Time*:

public void Check(int seconds, int minutes, int hours)

{

if (seconds < 60)

{

\_seconds += seconds;

}

else

{

seconds -= 60;

\_seconds += seconds;

\_minutes++;

}

if (minutes < 60)

{

\_minutes += minutes;

}

else

{

minutes -= 60;

\_minutes += minutes;

\_hours++;

}

\_hours += hours;

}

Метод *Check*, що виконує перевірку правильності запису дати у нормальному форматі. Використовується в статичному методі *Parse* та операторі віднімання:

public void Check()

{

if(\_seconds>60)

{

\_seconds -= 60;

\_minutes++;

}

if(\_minutes>60)

{

\_minutes -= 60;

\_hours++;

}

if(\_seconds<0)

{

\_seconds += 60;

\_minutes--;

}

if(\_minutes<0)

{

\_minutes += 60;

\_hours--;

}

if(\_hours<0)

{

Console.WriteLine("Error");

}

}

Перевантаження оператору зменшення для виконання дії віднімання між двома екземплярами класу *Time*:

public static Time operator - (Time time\_start, Time time\_finish)

{

time\_start.\_seconds -= time\_finish.\_seconds;

time\_start.\_minutes -= time\_finish.\_minutes;

time\_start.\_hours -= time\_finish.\_hours;

time\_start.Check();

return time\_start;

}

Статичний метод *Parse* класу *Time*, що розділяє строку на екземпляр класу:

public static Time Parse(string time)

{

string[] str = time.Split(' ');

Time t = new Time();

if(str.Length == 1)

{

int seconds = int.Parse(str[0]);

t.\_hours = seconds / 60 / 60;

t.\_minutes = (seconds / 60) - (t.\_hours \* 60);

t.\_seconds = seconds - ((t.\_minutes \* 60) + (t.\_hours \* 60 \* 60));

t.Check();

return t;

}

t.\_seconds = int.Parse(str[0]);

t.\_minutes = int.Parse(str[1]);

t.\_hours = int.Parse(str[2]);

t.Check();

return t;

}

Перевизначення статичного методу ToString базового класу Object:

public override string ToString()

{

return "Hours: "+\_hours+" Minutes: "+\_minutes+" Seconds: "+\_seconds;

}

Деструктор класу *Time*:

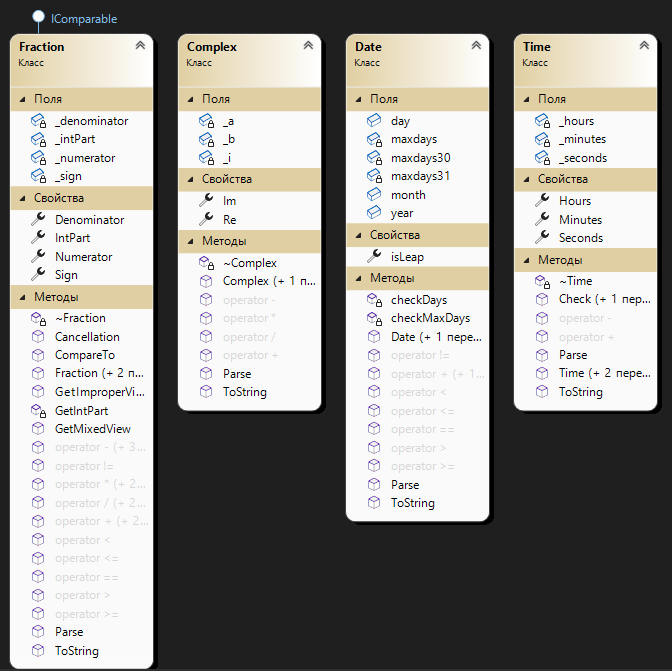
~Time()//деструктор

{

Console.WriteLine("Object has destroyed.");

}

1. UML-діаграма класів:



1. Демонстрація роботи класів.

3.1. Робота класу *Fraction*:

1)Перетворення десяткового дробу у звичайний із виділенням цілої частини:

using System;

namespace prac1\_console

{

internal class Program

{

static void Main(string[] args)

{

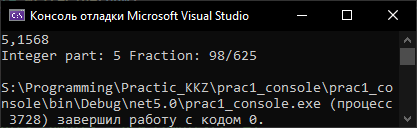
Fraction fraction1 = new Fraction(5.1568);

Console.WriteLine(fraction1);

}

}

}



2)Додавання двох дробів:

using System;

namespace prac1\_console

{

internal class Program

{

static void Main(string[] args)

{

Fraction fraction1 = new Fraction(25, 50);

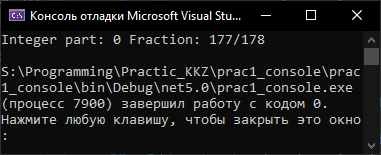
Fraction fraction2 = new Fraction(44, 89);

Console.WriteLine(fraction1 + fraction2);

}

}

}



3)Віднімання дробів:

using System;

namespace prac1\_console

{

internal class Program

{

static void Main(string[] args)

{

Fraction fraction1 = new Fraction(25, 50);

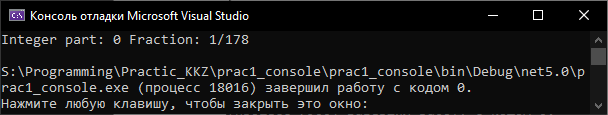
Fraction fraction2 = new Fraction(44, 89);

Console.WriteLine(fraction1 - fraction2);

}

}

}



3.2. Робота класу *Complex*:

1)Додавання двох комплексних чисел:

using System;

namespace prac1\_console

{

internal class Program

{

static void Main(string[] args)

{

Complex complex1 = new Complex(25, 50);

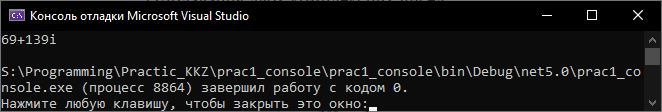
Complex complex2 = new Complex(44, 89);

Console.WriteLine(complex1 + complex2);

}

}

}



2)Ділення двох комплексних чисел:

using System;

namespace prac1\_console

{

internal class Program

{

static void Main(string[] args)

{

Complex complex1 = new Complex(2, 8);

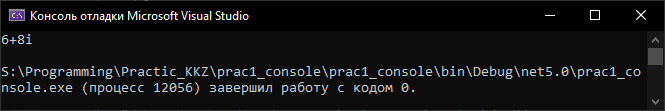
Complex complex2 = new Complex(4, 16);

Console.WriteLine(complex1 / complex2);

}

}

}



3.2. Робота класу *Date*:

1)Порівняння двох дат:

using System;

namespace prac1\_console

{

internal class Program

{

static void Main(string[] args)

{

Date date1 = new Date(25, 11, 2003);

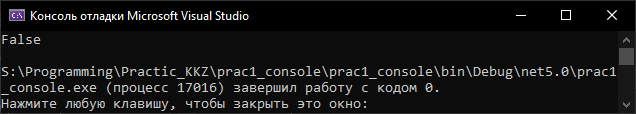
Date date2 = new Date(12, 12, 2005);

Console.WriteLine(date1 > date2);

}

}

}



2)Додавання днів до дати:

using System;

namespace prac1\_console

{

internal class Program

{

static void Main(string[] args)

{

Date date1 = new Date(25, 11, 2003);

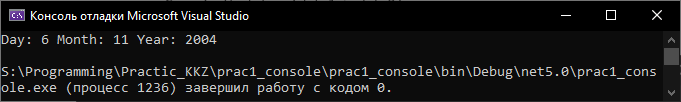
int days = 346;

Console.WriteLine(date1 + days);

}

}

}



3.3. Робота класу *Time*:

1)Додавання хвилин до часу:

using System;

namespace prac1\_console

{

internal class Program

{

static void Main(string[] args)

{

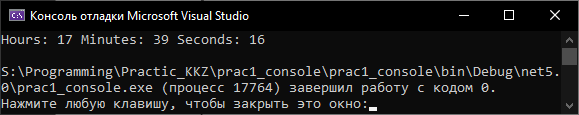
Time time1 = new Time(16, 52, 16);

int minutes = 47;

Console.WriteLine(time1 + minutes);

}

}}



2)Додавання хвилин до часу:

using System;

namespace prac1\_console

{

internal class Program

{

static void Main(string[] args)

{

Time time1 = new Time(16, 52, 16);

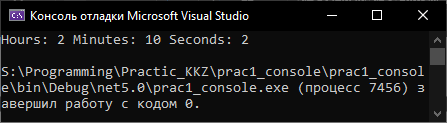
Time time2 = new Time(78, 121, 17);

Console.WriteLine(time2 - time1);

}

}

}



Лістинг класу *Program*:

using System;

namespace prac1\_console

{

internal class Program

{

static void Main(string[] args)

{

Fraction fraction10 = new Fraction(5.1568);

Console.WriteLine(fraction10 + "\n");

Fraction fraction1 = new Fraction(25, 50);

Fraction fraction2 = new Fraction(44, 89);

Console.WriteLine(fraction1 + fraction2 + "\n");

Fraction fraction3 = new Fraction(25, 50);

Fraction fraction4 = new Fraction(44, 89);

Console.WriteLine(fraction3 - fraction4 + "\n");

Complex complex1 = new Complex(25, 50);

Complex complex2 = new Complex(44, 89);

Console.WriteLine(complex1 + complex2 + "\n");

Complex complex3 = new Complex(2, 8);

Complex complex4 = new Complex(4, 16);

Console.WriteLine(complex3 / complex4 + "\n");

Date date1 = new Date(25, 11, 2003);

Date date2 = new Date(12, 12, 2005);

Console.WriteLine(date1 > date2);

Console.WriteLine("\n");

Date date3 = new Date(25, 11, 2003);

int days = 346;

Console.WriteLine(date3 + days + "\n");

Time time1 = new Time(16, 52, 16);

int minutes = 47;

Console.WriteLine(time1 + minutes + "\n");

Time time2 = new Time(16, 52, 16);

Time time3 = new Time(78, 121, 17);

Console.WriteLine(time3 - time2 + "\n");

}

}

}

**ПРАКТИЧНА РОБОТА №2**

**«ВИКОРИСТАННЯ КОЛЕКЦІЙ. КЛАСИ ARRAYLIST,QUEUE, STACK, HASHTABLE, SORTEDLIST»**

**ВАРІАНТ №3**

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

**Вправа 3:**

Повний лістинг програми:

using System;

namespace prac2\_console

{

internal class Program

{

static void task1(int[,] array)

{

bool rowHasNegative = false;

int product = 1;

int j = 0;

for (int i = 0; i < array.GetLength(0); i++)

{

while (j < array.GetLength(1) && !rowHasNegative)

{

if (array[i, j] < 0)

rowHasNegative = true;

else

product \*= array[i, j];

j++;

}

if (!rowHasNegative)

{

Console.WriteLine($"The product of row #{i+1}: {product}");

product = 1;

}

j = 0;

rowHasNegative = false;

}

}

static void task2(int[,] array)

{

int maxLength = 0;

int maxNumber = 0;

int Row = 0;

int currentNumber;

int currentLength = 0;

for (int i = 0; i < array.GetLength(0); i++)

{

for (int j = 0; j < array.GetLength(1); j++)

{

currentNumber = array[i,j];

for (int k = 0; k < 3; k++)

{

if (currentNumber == array[i,k])

{

currentLength++;

}

}

if (currentLength>maxLength)

{

maxLength = currentLength;

maxNumber = array[i, j];

Row = i + 1;

}

currentLength = 0;

}

}

Console.WriteLine($"Row with maximum sequence is #{Row}. Max squence is {maxLength}. Max number is {maxNumber}");

}

static void Main(string[] args)

{

int[,] array = { { 1, -3, 5 }, { 2, 1, 1 }, { 4, 4, 9 } };

task1(array);

task2(array);

}

}

}

Програма складається з класу Program в якому є 3 методи: основний метод Main та два методи task1, task2 що виконують дії, поставлені в завданні.

У статичному методі *Main* створюється та ініціалізується двомірний масив, викликаються 2 інших методи, в які передається попередньостворений масив:

static void Main(string[] args)

{

int[,] array = { { 1, -3, 5 }, { 2, 1, 1 }, { 4, 4, 9 } };

task1(array);

task2(array);

}

Статичний метод *task1* за допомогою циклу перевіряє кожен рядок на наявність негативних елементів, якщо таких намає, то перемножує едементи цього рядка і виводить на консоль номер рядка та добуток його елементів:

static void task1(int[,] array)

{

bool rowHasNegative = false;

int product = 1;

int j = 0;

for (int i = 0; i < array.GetLength(0); i++)

{

while (j < array.GetLength(1) && !rowHasNegative)

{

if (array[i, j] < 0)

rowHasNegative = true;

else

product \*= array[i, j];

j++;

}

if (!rowHasNegative)

{

Console.WriteLine($"The product of row #{i+1}: {product}");

product = 1;

}

j = 0;

rowHasNegative = false;

}

}

Статичний метод *task2* також за допомогою циклу проходить кожний рядок та в змінну maxLength записує максимальне значення серії та номер рядка, в якому ця серія знаходиться. Також, в змінну maxNumber записується максимальне значення цього рядка.

static void task2(int[,] array)

{

int maxLength = 0;

int maxNumber = 0;

int Row = 0;

int currentNumber;

int currentLength = 0;

for (int i = 0; i < array.GetLength(0); i++)

{

for (int j = 0; j < array.GetLength(1); j++)

{

currentNumber = array[i,j];

for (int k = 0; k < 3; k++)

{

if (currentNumber == array[i,k])

{

currentLength++;

}

}

if (currentLength>maxLength)

{

maxLength = currentLength;

maxNumber = array[i, j];

Row = i + 1;

}

currentLength = 0;

}

}

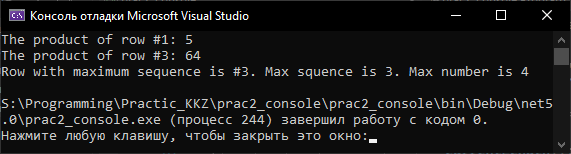


Рисунок 1 – демонстрація роботи програми

**Вправа 1.** Опис роботи класу ArrayList.

Лістинг до програми:

using System;

using System.Collections;

namespace ArrayList2

{

class Program

{

static void Main(string[] args)

{

ArrayList baseballTeams = new ArrayList();

baseballTeams.Add("St. Louis Cardinals");

baseballTeams.Add("Seattle Mariners");

baseballTeams.Add("Florida Marlins");

string[] myStringArray = new string[2];

myStringArray[0] = "San Francisco Giants";

myStringArray[1] = "Los Angeles Dodgers";

baseballTeams.AddRange(myStringArray);

foreach (string item in baseballTeams)

{

Console.Write(item + "\n");

}

Console.ReadLine();

}

}

}

В ході виконання програми створюється колекція ArrayList яка представляє собою список баскетбольних команд та в неї додаються 3 команди. Колекція ArrayList може зберігати в собі об’єкти різних типів. Далі створюється строковий масив та в нього додаються ще дві команди. Використовуючи метод AddRange ми можемо об’єднати данні з інших масивів або колекцій у об’єкті ArrayList. Далі використовуючи цикл foreach виводимо дані з колекції на екран. Використати цей цикл ми можем через те, що всі базові колекції поля імен System.Collections реалізують інтерфейс IEnumerable, що дозволяє інтексувати усі елементи колекцій. На рисунку 2 показаний результат роботи програми.

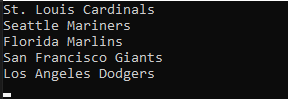


Рисунок 2 – результат виконання програми.

**Вправа 2.** Опис роботи HashTable

Лістинг до програми:

using System;

using System.Text;

using System.Collections;

namespace SocialSecurityNumbers

{

class MainEntryPoint

{

static void Main()

{

TestHarness harness = new TestHarness();

harness.Run();

}

}

class TestHarness

{

Hashtable employees = new Hashtable(31);

public void Run()

{

EMPLOYEEID idMortimer = new EMPLOYEEID("B001");

EmployeeData mortimer = new EmployeeData(idMortimer, "Mortimer", 100000.00M);

EMPLOYEEID idArabel = new EMPLOYEEID("W234");

EmployeeData arabel = new EmployeeData(idArabel, "Arabel Jones", 10000.00M);

employees.Add(idMortimer, mortimer);

employees.Add(idArabel, arabel);

while (true)

{

try

{

Console.Write("Enter employee ID (format:A999, X to exit)> ");

string userInput = Console.ReadLine();

userInput = userInput.ToUpper();

if (userInput == "X")

return;

EMPLOYEEID id = new EMPLOYEEID(userInput);

DisplayData(id);

}

catch (Exception e)

{

Console.WriteLine("Exception occurred. Did you use the correct format for the employee ID?");

Console.WriteLine(e.Message);

Console.WriteLine();

}

Console.WriteLine();

}

}

private void DisplayData(EMPLOYEEID id)

{

object empobj = employees[id];

if (empobj != null)

{

EmployeeData employee = (EmployeeData)empobj;

Console.WriteLine("Employee: " + employee.ToString());

}

else

Console.WriteLine("Employee not found: ID = " + id);

}

}

class EmployeeData

{

private string name;

private decimal salary;

private EMPLOYEEID id;

public EmployeeData(EMPLOYEEID id, string name, decimal salary)

{

this.id = id;

this.name = name;

this.salary = salary;

}

public override string ToString()

{

StringBuilder sb = new StringBuilder(id.ToString(), 100);

sb.Append(": ");

sb.Append(string.Format("{0-20}", name));

sb.Append(" ");

sb.Append(string.Format("{0:C}", salary));

return sb.ToString();

}

}

class EMPLOYEEID

{

private readonly char prefix;

private readonly int number;

public EMPLOYEEID(string id)

{

prefix = (id.ToUpper())[0];

number = int.Parse(id.Substring(1, 3));

}

public override string ToString()

{

return prefix.ToString() + string.Format("{0,3:000}", number);

}

public override int GetHashCode()

{

string str = this.ToString();

return str.GetHashCode();

}

public override bool Equals(object obj)

{

EMPLOYEEID rhs = obj as EMPLOYEEID;

if (rhs == null)

return false;

if (prefix == rhs.prefix && number == rhs.number)

return true;

return false;

}

}

}

У методі Main створюється еземпляр класу TestHarness та одразу викликається метод Run(), що ініціалізує роботу програми. У класі TestHarnes також був створений об’єкт класу колекції Hashtable та ініціалізований конструктором що приймає цілочислену змінну, що задає вмістимість колекції.

На початку методу Run() створюється еземпляр класу EMPLOYEEID та EmployeeData.

Лістинг цього класу:

class EMPLOYEEID

{

private readonly char prefix;

private readonly int number;

public EMPLOYEEID(string id)

{

prefix = (id.ToUpper())[0];

number = int.Parse(id.Substring(1, 3));

}

public override string ToString()

{

return prefix.ToString() + string.Format("{0,3:000}", number);

}

public override int GetHashCode()

{

string str = this.ToString();

return str.GetHashCode();

}

public override bool Equals(object obj)

{

EMPLOYEEID rhs = obj as EMPLOYEEID;

if (rhs == null)

return false;

if (prefix == rhs.prefix && number == rhs.number)

return true;

return false;

}

}

В цьому класі було створено 2 змінні prefix та number. Був створений конструктор з одним строковим параметром, що задає id робітника. Перший символ повинен бути символом, що відповідає латинському алфавіту та 3 цифри, що задають числову частину id. Потім був перевизначені усі методи, базового для всіх об’єктів, класу object. Метод ToString() задає правило перетворення еземплярів у строку, його перевизначення потрібне для зручного виводу потрібної інформації на екран. Метод Equals(object obj) який визначає правила порівняння 2 екземплярів між собою. Метод GetHashCode визначає правило отримання хеш коду для об’єктів. Хеш код це своєрідний ідентифікатор для об’єктів у цій програмі він потрібен для використання колекції Hashtable, бо вона використовує пари ключ/значення, а ці коди представляють собою ключі.

Далі був створений клас EmployeeData. В якому створені приватні поля імені, заробітної плати та шляхом агрегації реалізован об’єкт класу EMPLOYEEID. Також перевизначений метод ToString в якому використовуючи клас StringBuilder, що представляє динамічну строку, виведені данні про працівника на екран.

Створивши екземпляри цих класів у методі Run класу TestHarness, вони заносяться у колекцію Hashtable за правилом ключ – значення, де ключ це EMPLOYEEID, а значення це EmployeeData. Далі створюється цикл while та у конструкції try-catch Користувачу надається можливість здійснити пошук серед доданих у колекцію елементів шляхом введення id цих робітників. При введені некоректного значення id буде сгенерована помилка та її текст буде виведений на екран. Для того щоб вийти з циклу користувачу потрібно ввести «Х». На рисунку 3 показаний результат роботи програми.

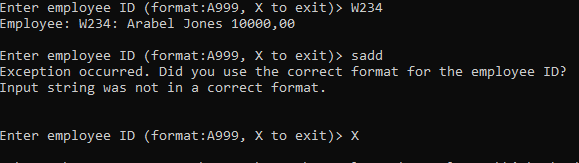


Рисунок 3 – результат виконання програми.

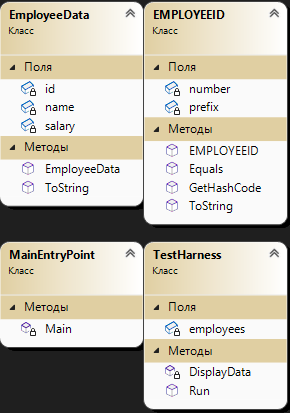


Рисунок 4 – UML діаграма класів

**ПРАКТИЧНА РОБОТА №3**

**«КЛАСИ В C# ТА СПАДКУВАННЯ»**

**ВАРІАНТ №5**

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

**Вправа 3:**

3.1. Перелік класів: Організація, Страхова компанія, Суднобудівельна компанія, Завод.

Повний лістинг програми:

using System;

using System.Collections.Generic;

namespace prac3\_console

{

public abstract class Organization

{

private string \_name;

private string \_industry;

private string \_area\_served;

public static LinkedList<Organization> List = new LinkedList<Organization>();//контрваріантність, в колекцію на основі базового класу можна добавти класи наслідник

public static void CheckList()

{

foreach (var item in List)

{

item.GetInfo();

}

}

public string Name

{

get { return \_name; }

set { \_name = value; }

}

public string Industry

{

get { return \_industry; }

set { \_industry = value; }

}

public string Area\_served

{

get { return \_area\_served; }

set { \_area\_served = value; }

}

public abstract void GetInfo();

public void Add()

{

List.AddLast(this);

}

public Organization()

{

\_name = "No name";

\_industry = null;

\_area\_served = null;

}

public Organization(string name, string industry, string area\_served)

{

\_name = name;

\_industry = industry;

\_area\_served = area\_served;

}

~Organization()

{

Console.WriteLine("Desposing of" + \_name);

}

}

public class Factory : Organization

{

private string \_head;

private int \_number\_of\_employees;

private string \_city;

public string Head

{

get { return \_head; }

set { \_head = value; }

}

public int N\_O\_E

{

get { return \_number\_of\_employees; }

set { \_number\_of\_employees = value; }

}

public string City

{

get { return \_city; }

set { \_city = value; }

}

public override void GetInfo()

{

Console.WriteLine($"Factory: \n\tName of organisation: {Name}");

Console.WriteLine("\tName: {0}\n\tHead: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

Name, \_head, \_number\_of\_employees, \_city);

}

public Factory() : base()

{

\_head = null;

\_number\_of\_employees = 0;

\_city = null;

}

public Factory(string name, string industry, string area\_served, string head, string city, int number\_of\_employees) : base(name, industry, area\_served)

{

\_head = head;

\_city = city;

\_number\_of\_employees = number\_of\_employees;

}

~Factory() { }

}

public class Insurance : Organization

{

private int \_client\_number;

private string \_speciality;

public int CN

{

get { return \_client\_number; }

set { \_client\_number = value; }

}

public string Speciality

{

get { return \_speciality; }

set { \_speciality = value; }

}

public override void GetInfo()

{

Console.WriteLine($"Insurance: \n\tName of organisation: {Name}");

Console.WriteLine("\tNumber of clients: {0}\n\tSpecialization: {1}\n", \_client\_number, \_speciality);

}

public Insurance() : base() { }

public Insurance(string name, string industry, string area\_served,

int client\_number, string spec) : base(name, industry, area\_served)

{

this.\_client\_number = client\_number;

this.\_speciality = spec;

}

~Insurance() { }

}

public class BuildingCompany : Organization

{

private string \_speciality;

private string \_city;

private int \_number\_of\_employees;

public string Speciality { set => \_speciality = value; get => \_speciality; }

public string City { set => \_city = value; get => \_city; }

public int N\_O\_E { set => \_number\_of\_employees = value; get => \_number\_of\_employees; }

public override void GetInfo()

{

Console.WriteLine($"Building Company: \n\tName of organisation: {Name}");

Console.WriteLine("\tName: {0}\n\tSpeciality: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

Name, \_speciality, \_number\_of\_employees, \_city);

}

public BuildingCompany() { }

public BuildingCompany(string name, string industry, string area\_wide, string spec, string city, int n\_o\_e) : base(name, industry, area\_wide)

{

}

~BuildingCompany() { }

}

internal class Program

{

static void Main(string[] args)

{

Factory f = new Factory("agrr", "oil", "Country", "Popolski", "Kyiv", 900);

f.GetInfo();

BuildingCompany bc = new BuildingCompany("Building Co.", "skys", "City", "Gladkevich", "Odesa", 1900);

Insurance ins = new Insurance("Insurance Co. ", "Money", "City",300, "Cars");

Organization.CheckList();

f.Add();

bc.Add();

ins.Add();

Organization.CheckList();

}

}

}

Ієрархія класів, визначена у програмі:

Клас-батько: *Organiztion*

Класи-спадкоємці від *Organization*: *BuildingCompany, Factory, Insurance.*

3.2. Опис абстрактного класу *Organiztion:*

public abstract class Organization

{

private string \_name;

private string \_industry;

private string \_area\_served;

public static LinkedList<Organization> List = new LinkedList<Organization>();

public static void CheckList()

{

foreach (var item in List)

{

item.GetInfo();

}

}

public string Name

{

get { return \_name; }

set { \_name = value; }

}

public string Industry

{

get { return \_industry; }

set { \_industry = value; }

}

public string Area\_served

{

get { return \_area\_served; }

set { \_area\_served = value; }

}

public abstract void GetInfo();

public void Add()

{

List.AddLast(this);

}

public Organization()

{

\_name = "No name";

\_industry = null;

\_area\_served = null;

}

public Organization(string name, string industry, string area\_served)

{

\_name = name;

\_industry = industry;

\_area\_served = area\_served;

}

~Organization()

{

Console.WriteLine("Desposing of" + \_name);

}

}

В класі визначені 2 конструктори, перший пустий, в другий передаються такі дані, як Назва, Індустрія та Область займаності компанії. Також визначений деструктор:

public Organization()

{

\_name = "No name";

\_industry = null;

\_area\_served = null;

}

public Organization(string name, string industry, string area\_served)

{

\_name = name;

\_industry = industry;

\_area\_served = area\_served;

}

~Organization()

{

Console.WriteLine("Desposing of" + \_name);

}

Визначаються декілька приватних змінних, проводиться їх інкапсуляція:

private string \_name;

private string \_industry;

private string \_area\_served;

public string Name

{

get { return \_name; }

set { \_name = value; }

}

public string Industry

{

get { return \_industry; }

set { \_industry = value; }

}

public string Area\_served

{

get { return \_area\_served; }

set { \_area\_served = value; }

}

Створення об’єкту колекції *LinkedList* на основі класу *Organization* та метод CheckList , що виводить всі елементи списку:

public static LinkedList<Organization> List = new LinkedList<Organization>();

public static void CheckList()

{

foreach (var item in List)

{

item.GetInfo();

}

}

Метод *Add* виконує функцію додавання елементів у кінець двозв’язанного списку *List*:

public void Add()

{

List.AddLast(this);

}

3.3. Опис класу-спадкоємцю *Factory:*

public class Factory : Organization

{

private string \_head;

private int \_number\_of\_employees;

private string \_city;

public string Head

{

get { return \_head; }

set { \_head = value; }

}

public int N\_O\_E

{

get { return \_number\_of\_employees; }

set { \_number\_of\_employees = value; }

}

public string City

{

get { return \_city; }

set { \_city = value; }

}

public override void GetInfo()

{

Console.WriteLine($"Factory: \n\tName of organisation: {Name}");

Console.WriteLine("\tName: {0}\n\tHead: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

Name, \_head, \_number\_of\_employees, \_city);

}

public Factory() : base()

{

\_head = null;

\_number\_of\_employees = 0;

\_city = null;

}

public Factory(string name, string industry, string area\_served, string head, string city, int number\_of\_employees) : base(name, industry, area\_served)

{

\_head = head;

\_city = city;

\_number\_of\_employees = number\_of\_employees;

}

~Factory() { }

}

Визначення приватних змінних та присвоєння їм властивостей:

private string \_head;

private int \_number\_of\_employees;

private string \_city;

public string Head

{

get { return \_head; }

set { \_head = value; }

}

public int N\_O\_E

{

get { return \_number\_of\_employees; }

set { \_number\_of\_employees = value; }

}

public string City

{

get { return \_city; }

set { \_city = value; }

}

Опис двох конструкторів та деструктора класу *Factory*. Перший конструктор нічого не приймає, в другий передаються змінні string name, string industry, string area\_served, string head, string city, int number\_of\_employees серед яких name, industry, та area\_served наслідуються від базового класу яким є *Organization.*

public Factory() : base()

{

\_head = null;

\_number\_of\_employees = 0;

\_city = null;

}

public Factory(string name, string industry, string area\_served, string head, string city, int number\_of\_employees) : base(name, industry, area\_served)

{

\_head = head;

\_city = city;

\_number\_of\_employees = number\_of\_employees;

}

~Factory() { }

}

Метод GetInfo виводить у консоль інформацію про поточну організацію:

public override void GetInfo()

{

Console.WriteLine($"Factory: \n\tName of organisation: {Name}");

Console.WriteLine("\tName: {0}\n\tHead: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

Name, \_head, \_number\_of\_employees, \_city);

}

Класи *BuildingCompany, Insurance* мають таку ж реалізацію як і клас *Factory.*



Рисунок 3.1 - UML-діаграма.

3.5. Демонстрація роботи проекту:

У статичному методі *Main* ініціалізуються екземпляри класів-спадкоємців. За допомогою методу *Add* класу *Organization* екземпляри класів-спадкоємців додаються до списку *List*(Рисунок 3.2).

static void Main(string[] args)

{

Factory f = new Factory("agrr", "oil", "Country", "Popolski", "Kyiv", 900);

f.GetInfo();

BuildingCompany bc = new BuildingCompany("Building Co.", "skys", "City", "Gladkevich", "Odesa", 1900);

Insurance ins = new Insurance("Insurance Co. ", "Money", "City",300, "Cars");

Organization.CheckList();

f.Add();

bc.Add();

ins.Add();

Organization.CheckList();

}

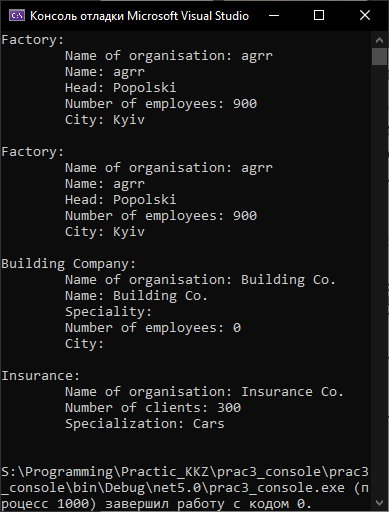


Рисунок 3.2 – результат роботи програми.

Вправа 1.

Лістинг до вправи:

using System;

using System.Threading;

class Car

{

private String mName;

public Car()

{

mName = "No Name";

}

public Car(String name)

{

mName = name;

}

public void Print()

{

Console.WriteLine("Name : " + mName);

}

~Car()

{

Console.WriteLine("Disposing of " + mName);

}

}

class TestMain

{

public static void Main()

{

for (int i = 0; i < 9000; ++i)

{

Car a = new Car("Name:" + i);

a.Print();

}

}

}

В ході виконання програми у циклі створюється велика кількість екземплярів класу та виводяться на екран їх імена далі після їх створення повинні виводитися повідомлення від деструктора, але виходячи з правил за якими працює Garbage Collector программа завершує свою роботу раніше чим встигнуть вивестися повідомлення деструктора.

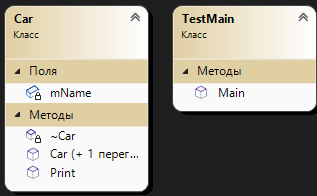


Рисунок 4 – UML діаграма класів

Вправа 2.1.

Повний лістинг:

using System;

public class Base

{

public void PrintItem()

{

for (int i = 0; i < 10; ++i)

Console.WriteLine("Item: " + this.GetItem(i));

}

public virtual int GetItem(int index)

{

return 0;

}

}

public class Derived : Base

{

public override int GetItem(int index)

{

return index \* index;

}

}

public class TestMain

{

public static void Main()

{

new Derived().PrintItem();

}

}

Напочатку створений клас Base в якому створені 2 метода який виводить результат роботи віртуального методу GetItem. У класі Derived, який успадковується від класу Base, далі перевизначається метод GetItem. У методі Main створюється об’єкт Derived та ініціалізується метод PrintItem. Результат роботи програми показан на рисунку 5.

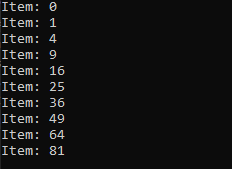


Рисунок 5 – Результат виконання програми

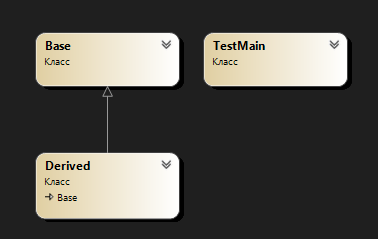


Рисунок 6 – UML діаграма класів

Вправа 2.2.

У цій вправі єдиною різницею з попередньою є в тому що метод GetItem є абстрактним. А тому в базовому калсі його тіло не створено, тому він має бути обов’язково перевизначений у похідному.

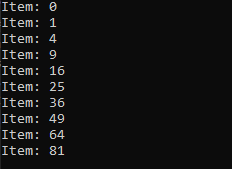


Рисунок 7 – Результат виконання програми

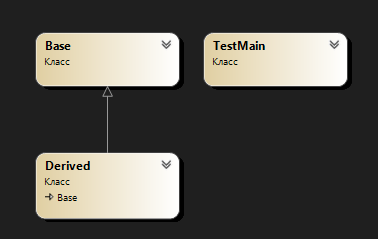


Рисунок 8 – UML діаграма класів

**ПРАКТИЧНА РОБОТА № 4**

**«ОБРОБКА ВИКЛЮЧЕНЬ В МОВІ C#»**

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

**Вправа 1.**

using System;

class ExcDemo1

{

public static void Main()

{

int[] nums = new int[4];

try

{

Console.WriteLine("Перед генеруванням виключення.");

for (int i = 0; i < 10; i++)

{

nums[i] = i;

Console.WriteLine("nums[{0}]: {1}", i, nums[i]);

}

Console.WriteLine("Цей текст не відображається.");

}

catch (IndexOutOfRangeException)

{

Console.WriteLine("Індекс поза діапазоном!");

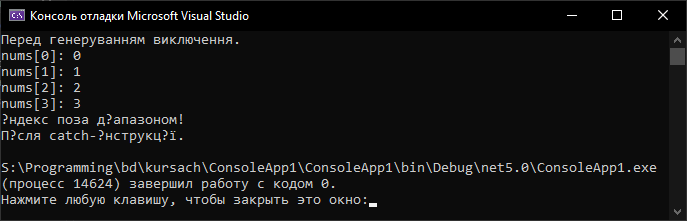
}

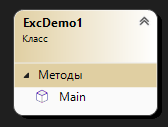
Console.WriteLine("Після catch-інструкції.");

}

}

Програма перехоплює вихід чисел за границю діапазона за допомогою обробника виняткових подій try-catch.





**Вправа 2.**

using System;

namespace AdvancedCSharp

{

public class MainEntryPoint

{

public static void Main()

{

string userInput; while (true)

{

try

{

Console.Write("Input а number between 0 and 5 " + "(or just hit return to exit)> ");

userInput = Console.ReadLine();

if (userInput == "") break;

int index = Convert.ToInt32(userInput);

if (index < 0 || index > 5)

throw new IndexOutOfRangeException("You typed in " + userInput); Console.WriteLine("Your number was " + index);

}

catch (IndexOutOfRangeException e)

{

Console.WriteLine("Exception: " + "Number should be between 0 and 5. " + e.Message);

}

catch (Exception e)

{

Console.WriteLine("An exception was thrown. Message was " + e.Message);

}

catch

{

Console.WriteLine("Some other exception has occurred");

}

finally

{

Console.WriteLine("Thank you");

}

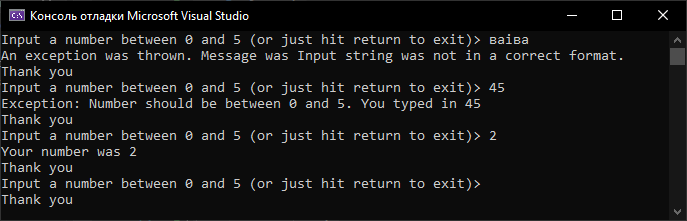
}

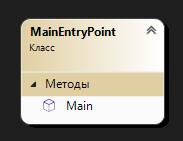
}

}

}

За допомогою блоків *try-catch-finally* програма перевіряє, чи входить у допустимий діапазон число, яке ввів користувач. Якщо не входить, виконається один з блоків *catch*.

****

****

**Завдання.**

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

* 1. Лістинг *Form1.cs*:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace prac4\_wf

{

public partial class Form1 : Form

{

string operation = "(pick operation)";

public Form1()

{

InitializeComponent();

}

private void button2\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void button1\_Click(object sender, EventArgs e)

{

if (radioButton1.Checked == true)

{

textBox4.Text = Convert.ToString(Convert.ToDouble(textBox1.Text) + Convert.ToDouble(textBox2.Text));

}

else if (radioButton2.Checked == true)

{

textBox4.Text = Convert.ToString(Convert.ToDouble(textBox1.Text) - Convert.ToDouble(textBox2.Text));

}

else if (radioButton3.Checked == true)

{

textBox4.Text = Convert.ToString(Convert.ToDouble(textBox1.Text) \* Convert.ToDouble(textBox2.Text));

}

else if (radioButton5.Checked == true)

{

textBox4.Text = Convert.ToString(Convert.ToDouble(textBox1.Text) % Convert.ToDouble(textBox2.Text));

}

else if (radioButton4.Checked == true)

{

try

{

if (Convert.ToDouble(textBox2.Text) == 0)

{

throw new DivideByZeroException();

}

else

{

textBox4.Text = Convert.ToString(Convert.ToDouble(textBox1.Text) / Convert.ToDouble(textBox2.Text));

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

else

{

MessageBox.Show("Pick operation!");

}

}

private void textBox2\_TextChanged(object sender, EventArgs e)

{

textBox3.Text = "";

textBox3.Text = textBox1.Text + " " + operation + " " + textBox2.Text;

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

textBox3.Text = "";

textBox3.Text = textBox1.Text + " " + operation + " " + textBox2.Text;

}

private void radioButton1\_Click(object sender, EventArgs e)

{

operation = "+";

textBox3.Text = "";

textBox3.Text = textBox1.Text + " " + operation + " " + textBox2.Text;

}

private void radioButton2\_Click(object sender, EventArgs e)

{

operation = "-";

textBox3.Text = "";

textBox3.Text = textBox1.Text + " " + operation + " " + textBox2.Text;

}

private void radioButton3\_Click(object sender, EventArgs e)

{

operation = "\*";

textBox3.Text = "";

textBox3.Text = textBox1.Text + " " + operation + " " + textBox2.Text;

}

private void radioButton4\_Click(object sender, EventArgs e)

{

operation = "/";

textBox3.Text = "";

textBox3.Text = textBox1.Text + " " + operation+ " " + textBox2.Text;

}

private void radioButton5\_Click(object sender, EventArgs e)

{

operation = "%";

textBox3.Text = "";

textBox3.Text = textBox1.Text + " " + operation + " " + textBox2.Text;

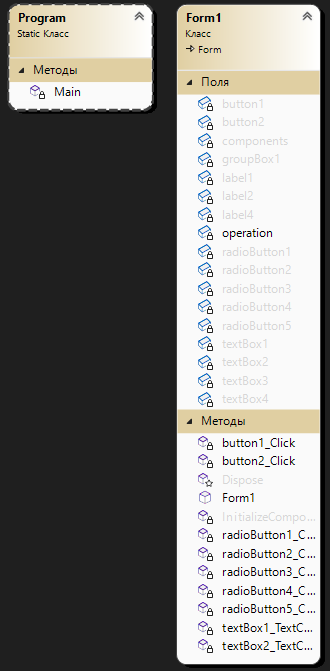
}

}

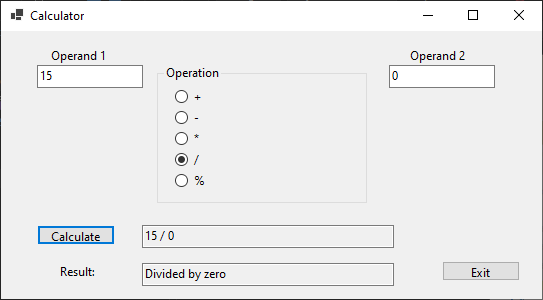
}

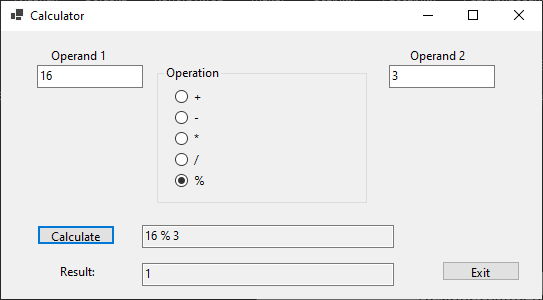
Клас *Form1,* що є класом-наслідником від базового класу *Form* має опис методів для виконання дій над числами в калькуляторі. Методи radioButton1\_Click, radioButton2\_Click, radioButton3\_Click, radioButton4\_Click, radioButton5\_Click є описом перемикачів, що визначають, яка дія має виконуватись. Метод button1\_Click описує кнопку «Calculate» та має в собі обробник виняткових ситуацій *try-catch*. У блоці *try* описані дії, під час яких може виникнути помилка(конвертація отриманих даних та виконання арифметичних дій), у блоці *catch* описаний виклик повідомлення про помилку, що виникла.

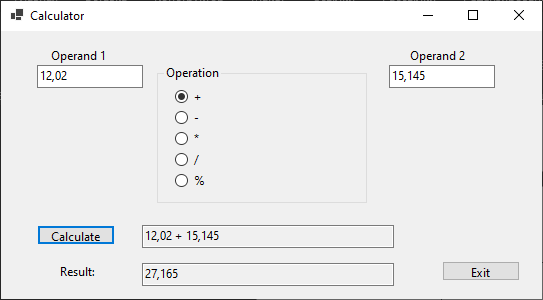
UML-діаграма класів:



Демонстрація роботи:







**ПРАКТИЧНА РОБОТА № 5**

**«АБСТРАКТНІ КЛАСИ»**

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

**Завдання 1.**

using System;

namespace prac5\_console

{

interface IPrint

{

public void Print();

}

abstract class GeometricShape

{

public abstract float GetArea();

}

class Rectangle: GeometricShape, IPrint

{

private float \_sideA, \_sideB;

public float SideA { get { return \_sideA; } set { \_sideA = value; } }

public float SideB { get { return \_sideB; } set { \_sideB = value; } }

public override float GetArea()

{

return \_sideA \* \_sideB;

}

public override string ToString()

{

return $"Side A: {\_sideA}\nSide B: {\_sideB}\nArea: {GetArea()}";

}

public void Print()

{

Console.WriteLine(this);

}

public Rectangle(float sideA, float sideB)

{

\_sideA = sideA;

\_sideB = sideB;

}

}

class Square: Rectangle

{

public override string ToString()

{

return $"Side: {SideA}\nArea: {GetArea()}";

}

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

{

}

}

class Circle: GeometricShape, IPrint

{

float \_radius;

public float Radius { get { return \_radius; } set { \_radius = value; } }

public override float GetArea()

{

return (float)Math.PI \* \_radius \* \_radius;

}

public override string ToString()

{

return $"Radius: {Radius}\nArea: {GetArea()}";

}

public void Print()

{

Console.WriteLine(this);

}

public Circle(float radius)

{

\_radius = radius;

}

}

internal class Program

{

static void Main(string[] args)

{

Circle circle= new Circle(4.6f);

circle.Print();

Console.WriteLine();

Rectangle rectangle = new Rectangle(4, 5);

rectangle.Print();

Console.WriteLine();

Square square = new Square(8);

square.Print();

}

}

}

Розроблений інтерфейс IPrint , що містить метод Print().

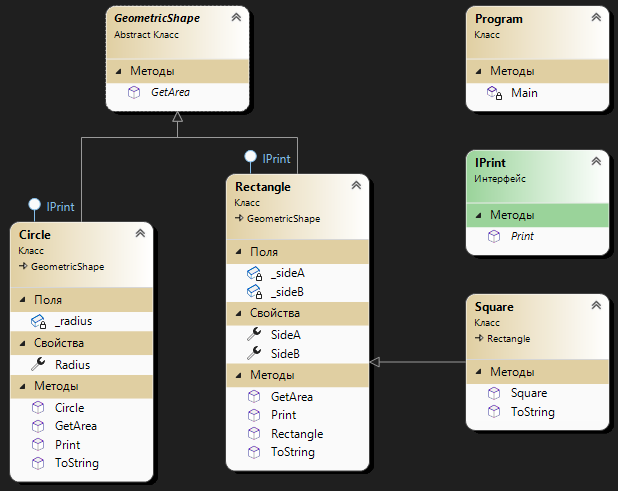
Програма має абстрактний клас GeometricShape , який містить віртуальний метод GetArea().

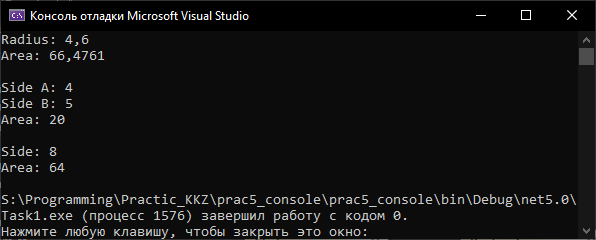
Клас Rectangle є спадкоємцем від класу GeometricShape. Змінні ширини та висоти оголошені як властивості(SideA, SideB), за цими параметрами створений конструктор public Rectangle.

Клас Square успадковується від класу Rectangle та має конструктор public Square, що приймає значення довжини сторони.

Клас Circle спадкоємець від GeometricShape. Має властивість Radius та конструтор public Circle за цією властивістю.

Для кожного класу перевизначено віртуальний метод ToString() базового класу Object, він повертає у вигляді рядка параметри фігури та її площу. Також, кожен клас реалізовує інтерфейс IPrint , мають перевизначуваний метод Print() , що виводить до консолі інформацію отриману, від методу ToString().





**Завдання 2.1**

using System;

namespace Task21

{

enum Shapes

{

Circle = 0,

Triangle = 3,

Square = 4

}

class Picture

{

Shape[] shapes;

int \_arraySize;

int \_index = 0;

public void Add(Shape shape)

{

try

{

shapes[\_index] = shape;

\_index++;

}

catch (Exception)

{

Console.WriteLine("You have reach the maximum number of elements");

}

}

public Picture(int arraySize)

{

\_arraySize = arraySize;

shapes = new Shape[arraySize];

}

public void PrintArray()

{

foreach (var item in shapes)

{

item.Print();

}

}

}

abstract class Shape

{

protected Shapes figure;

protected int sideLength;

public int SideLength { get { return sideLength; } }

public int VertexAmount { get; protected set; }

public abstract Shapes Figure { get; protected set; }

public abstract double GetArea();

public abstract double GetPerimeter();

public void Print()

{

Console.WriteLine("This is: " + Figure + "\n" + ToString());

}

public override string ToString()

{

return "Its side lenght: " + sideLength + "\nIts area: " + GetArea() + "\nIts perimeter: " + GetPerimeter() + "\n";

}

}

class Square : Shape

{

public override Shapes Figure

{

get { return figure; }

protected set { figure = value; }

}

public override double GetArea()

{

return sideLength\*sideLength;

}

public override double GetPerimeter()

{

return sideLength\*4;

}

public Square(int \_sideLenth)

{

VertexAmount = 4;

Figure = (Shapes)VertexAmount;

sideLength = \_sideLenth;

}

}

class Circle : Shape

{

public override Shapes Figure

{

get { return figure; }

protected set { figure = value; }

}

public override double GetArea()

{

return Math.PI\*sideLength\*sideLength;

}

public override double GetPerimeter()

{

return 2\*Math.PI\*sideLength;

}

public Circle(int \_sideLenth)

{

VertexAmount = 0;

Figure = (Shapes)VertexAmount;

sideLength = \_sideLenth;

}

public override string ToString()

{

return "Its radius: " + sideLength

+ "\nIts area: " + String.Format("{0:0.00}", GetArea())

+ "\nIts perimeter: " + String.Format("{0:0.00}", GetPerimeter())

+ "\n";

}

}

class Triangle : Shape

{

public override Shapes Figure

{

get { return figure; }

protected set { figure = value; }

}

public override double GetArea()

{

return (sideLength\*sideLength)/2;

}

public override double GetPerimeter()

{

return 3 \* sideLength;

}

public Triangle(int \_sideLenth)

{

VertexAmount = 3;

Figure = (Shapes)VertexAmount;

sideLength = \_sideLenth;

}

}

internal class Program

{

static void Main(string[] args)

{

Picture picture = new Picture(3);

picture.Add(new Square(5));

picture.Add(new Triangle(6));

picture.Add(new Circle(7));

picture.PrintArray();

}

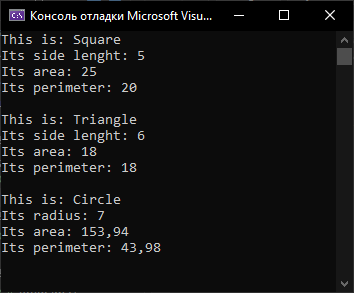
}

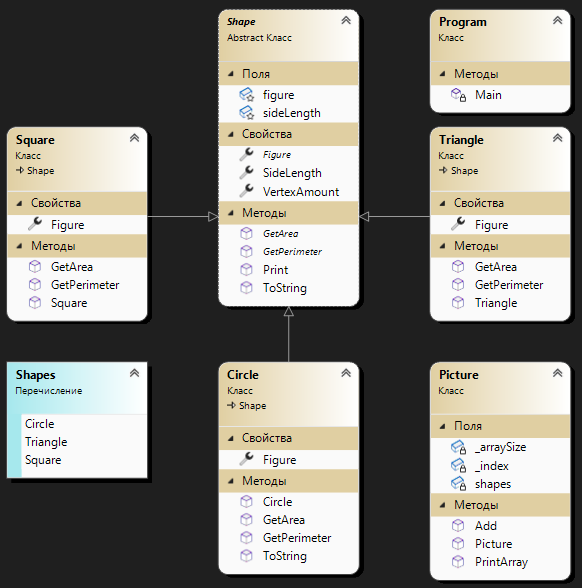
}

В програмі визначений абстракний клас *Shape,* з декількома абстрактними властивостями та абстрактинми методами. В ньому перевизначений метод *Tostring()* базового класу *Object.*

Визначені класи-спадкоємці від класу *Shape*: *Square, Circle, Triangle.* В кожному з цих класів реалізовані властивості та методи класу *Shape*.

Визначений клас *Picture,* що має конструктор, методи та властивості, що потребує завдання.





**Завдання 2.2**

using System;

namespace Task22

{

class Picture

{

Shape[] shapes;

int \_arraySize;

int \_index = 0;

public void Add(Shape shape)

{

try

{

shapes[\_index] = shape;

\_index++;

}

catch (Exception)

{

Console.WriteLine("You have reach the maximum number of elements");

}

}

public Picture(int arraySize)

{

\_arraySize = arraySize;

shapes = new Shape[\_arraySize];

}

public void PrintArray()

{

foreach (var item in shapes)

{

Console.WriteLine(item);

}

}

}

abstract class Shape

{

protected string figure;

protected float \_firstParam;

protected float \_secondParam;

public string Figure

{

get { return figure; }

protected set { figure = value; }

}

public abstract float FirstParam { get; set; }

public abstract float SecondParam { get; set; }

public abstract double GetArea();

public abstract double GetPerimeter();

public abstract void Print();

public override string ToString()

{

return "This is: " + Figure;

}

public Shape(float firstParam, float secondParam)

{

\_firstParam = firstParam;

\_secondParam = secondParam;

}

}

class Pentagon: Shape

{

public override float FirstParam { get => \_firstParam; set => \_firstParam = value; }

public override float SecondParam { get => \_secondParam; set => \_secondParam = value; }

public override double GetArea()

{

double height = (FirstParam / 2) / Math.Tan((36 \* Math.PI) / 180);

return 0.25 \* FirstParam \* height;

}

public override double GetPerimeter()

{

return FirstParam \* SecondParam;

}

public override void Print()

{

Console.WriteLine(ToString());

}

public override string ToString()

{

return base.ToString()

+ "\nIts side length: " + FirstParam

+ "\nIts area: " + String.Format("{0:0.00}", GetArea())

+ "\nIts perimeter: " + String.Format("{0:0.00}", GetPerimeter())

+ "\n";

}

public Pentagon(float firstParam): base(firstParam, 5)

{

Figure = "Pentagon";

}

}

class Ellipse: Shape

{

public override float FirstParam { get => \_firstParam; set => \_firstParam = value; }

public override float SecondParam { get => \_secondParam; set => \_secondParam = value; }

public override double GetArea()

{

return Math.PI \* FirstParam \* SecondParam;

}

public override double GetPerimeter()

{

return 4\*((Math.PI\*FirstParam\*SecondParam+(FirstParam-SecondParam))/(FirstParam+SecondParam));

}

public override void Print()

{

Console.WriteLine(ToString());

}

public override string ToString()

{

return base.ToString()

+ "\nIts long semiaxis: " + FirstParam

+ "\nIts short semiaxis: " + SecondParam

+ "\nIts area: " + String.Format("{0:0.00}", GetArea())

+ "\nIts perimeter: " + String.Format("{0:0.00}", GetPerimeter())

+ "\n";

}

public Ellipse(float firstParam, float secondParam) : base(firstParam, secondParam)

{

Figure = "Ellipse";

}

}

class Triangle: Shape

{

public override float FirstParam { get => \_firstParam; set => \_firstParam = value; }

public override float SecondParam { get => \_secondParam; set => \_secondParam = value; }

public override double GetArea()

{

return FirstParam \* SecondParam;

}

public override double GetPerimeter()

{

double g = Math.Sqrt(Math.Pow(\_firstParam,2)+Math.Pow(\_secondParam,2));

return FirstParam+SecondParam+g;

}

public override void Print()

{

Console.WriteLine(ToString());

}

public override string ToString()

{

return base.ToString()

+ "\nIts first leg: " + FirstParam

+ "\nIts second leg: " + SecondParam

+ "\nIts area: " + String.Format("{0:0.00}", GetArea())

+ "\nIts perimeter: " + String.Format("{0:0.00}", GetPerimeter())

+ "\n";

}

public Triangle(float firstParam, float secondParam) : base(firstParam, secondParam)

{

Figure = "Right triangle";

}

}

internal class Program

{

static void Main(string[] args)

{

Picture picture = new Picture(3);

picture.Add(new Pentagon(5));

picture.Add(new Triangle(6f,8f));

picture.Add(new Ellipse(4f,9f));

picture.PrintArray();

}

}

}

Визначений абстрактний клас *Shape* із абстрактними властивостями та методами.

Визначені класи-спадкоємці:

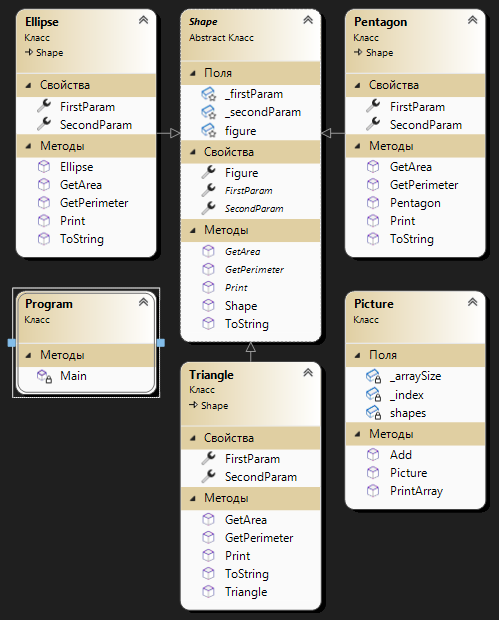
*Pentagon* (правильний п'ятикутник – задається довжиною сторони та числом сторін);

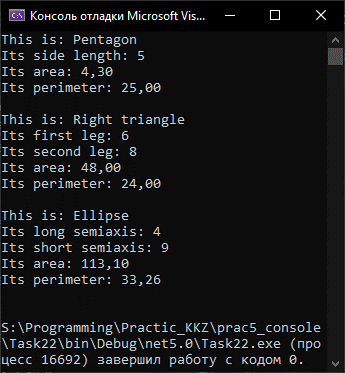
*Ellipse* (еліпс - задається пів осями);

*Triangle* (прямокутний трикутник – задається довжиною катетів).

У класах *Pentagon, Ellipse* та *Triangle* реалізовані абстрактні властивості та методи класу Shape.

Клас *Picture* визначений за зразком завдання 2.1.





**Завдання 2.3.**

using System;

namespace Task23

{

class Picture

{

Triangle[] triangles;

int \_arraySize;

int \_index = 0;

public void Add(Triangle triangle)

{

try

{

triangles[\_index] = triangle;

\_index++;

}

catch (Exception)

{

Console.WriteLine("You have reach the maximum number of elements");

}

}

public Picture(int arraySize)

{

\_arraySize = arraySize;

triangles = new Triangle[\_arraySize];

}

public void PrintArray()

{

foreach (var item in triangles)

{

Console.WriteLine(item);

}

}

}

abstract class Triangle

{

private string \_name;

public float \_firstSide;

public float \_secondSide;

public float \_angle;

public string Name { get => \_name; protected set => \_name = value; }

public abstract double GetArea();

public abstract double GetPerimeter();

public void Print()

{

Console.WriteLine(ToString());

}

public override string ToString()

{

return $"This is: {Name}\nIts first side: {\_firstSide}\nIts second side: {\_secondSide}\nIts angle: {\_angle}\nIts area: {GetArea()}\nIts peremiter: {GetPerimeter()}\n";

}

public Triangle(float firstSide, float secondSide, float angle)

{

\_firstSide = firstSide;

\_secondSide = secondSide;

\_angle = angle;

}

}

class EqTriangle: Triangle

{

public override double GetArea()

{

return (Math.Sqrt(3)/4)\*Math.Pow(\_firstSide, 2);

}

public override double GetPerimeter()

{

return \_firstSide\*3;

}

public override string ToString()

{

return $"This is: {Name}\nIts side: {\_firstSide}\nIts angle: {\_angle}\nIts area: {GetArea()}\nIts peremiter: {GetPerimeter()}\n";

}

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

{

Name = "EqTriangle";

}

}

class IsTriangle: Triangle

{

public override double GetArea()

{

double height = Math.Sin((\_angle \* Math.PI) / 180)\*\_firstSide;

return 0.5\*height\*\_secondSide;

}

public override double GetPerimeter()

{

return \_firstSide \* 2 + \_secondSide;

}

public IsTriangle(float firstSide, float secondSide, float angle) : base(firstSide, secondSide, angle)

{

Name = "IsTriangle";

}

}

class RectTriangle: Triangle

{

public override double GetArea()

{

return 0.5 \* \_secondSide \* \_firstSide;

}

public override double GetPerimeter()

{

double g = Math.Sqrt(Math.Pow(\_firstSide, 2) + Math.Pow(\_secondSide, 2));

return \_firstSide + \_secondSide + g;

}

public RectTriangle(float firstSide, float secondSide) : base(firstSide, secondSide, 90)

{

Name = "RectTriangle";

}

}

internal class Program

{

static void Main(string[] args)

{

Picture picture = new Picture(3);

picture.Add(new RectTriangle(5,9));

picture.Add(new IsTriangle(6f, 8f,35));

picture.Add(new EqTriangle(9f));

picture.PrintArray();

}

}

}

Визначений абстрактний клас *Triangle* з абстрактними властивостями та методами.

Визначені класи – спадкоємці від класу *Triangle*:

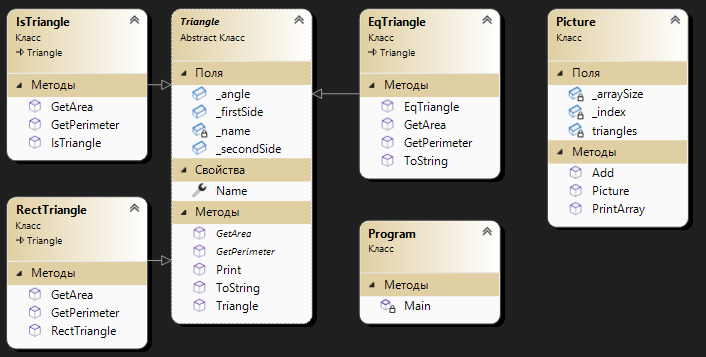
*EqTriangle* (рівносторонній трикутник);

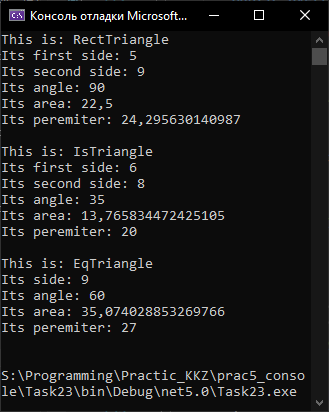
*IsTriangle* (рівнобедрений трикутник);

*RectTriangle* (прямокутний трикутник).

У класах *EqTriangle*, *IsTriangle* та *RectTriangle* реалізовані абстрактні властивості та методи класу Triangle.

Визначений клас *Picture*, який містить масив об'єктів цих класів.





**ПРАКТИЧНА РОБОТА № 6**

**«ВІРТУАЛЬНІ МЕТОДИ»**

**ВАРІАНТ №3**

**Мета роботи:** Навчитися застосовувати віртуальні класи при розробці проектів мовою С#.

Код програми:

using System;

namespace prac6\_console

{

internal class Program

{

class Harvesting

{

protected int \_area;

protected int \_per\_hectar;

public int Area { get => \_area; set => \_area = value; }

public int Per\_Hectar { get => \_per\_hectar; set => \_per\_hectar = value; }

public Harvesting()

{

\_area = 0;

\_per\_hectar = 0;

}

public Harvesting(int area)

{

\_area = area;

}

public virtual int Total\_Harvest()

{

return \_area \* \_per\_hectar;

}

public override string ToString()

{

return String.Format("Area: {0}\n Cost per hectar: {1}\n Total harvest cost: {2}", \_area ,\_per\_hectar, Total\_Harvest());

}

}

class Farm : Harvesting

{

public Farm(int area) : base(area)

{

\_per\_hectar = 120;

}

public override int Total\_Harvest()

{

int tax = 30;

return \_area \* (\_per\_hectar - tax) ;

}

}

class Homesteads : Harvesting

{

public Homesteads(int area) : base(area)

{

\_per\_hectar = 100;

}

public override int Total\_Harvest()

{

int tax = 22;

int expenses = \_area / 20;

return (\_area \* (\_per\_hectar - tax)) - expenses;

}

}

static void Main(string[] args)

{

Farm f = new Farm(500);

Homesteads h = new Homesteads(1200);

Console.WriteLine(f);

Console.WriteLine(h);

}

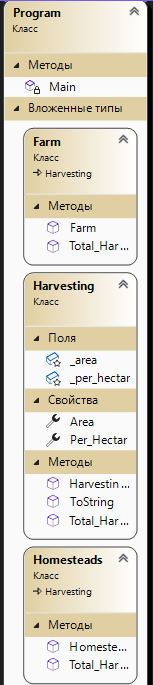
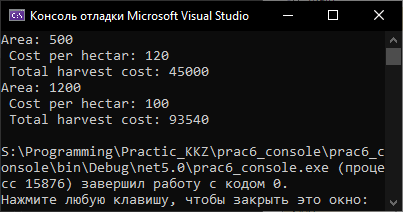
}

}

На початку програми оголошується базовий класс Harvesting який є описом аграрного підприємства або ділянки. Він має 2 приватні змінні що відповідають за розмір ділянки та кількість доступ до них відбувається через відповідні властивості. Також був створений віруальний метод Total\_Harvest(), що обчислює сумарний врожай шляхом множення площі ділянки (у гектарах) на кількість рослин на гектар, та перевизначений метод ToString() базового класу *Object*.

Потім були створені 2 класи спадкоємці. Перший - Farm що характеризує собою ферму. У ньому був перевизначений віртуальний метод шляхов додовання нового елементу tax (податок), так як ферми повинні сплачувати податок на свій дохід. Другий – Homesteads що характеризує присадибну ділянку в ньому також був перевизначений метод Total\_Harvest() додавши ще один показчик, а саме витрати на комуналні послуги, на приклад витрати на воду для поливу.

У методі Main створюються екземпляри класів нащадків та виводитсья інформація про них на екран.



**ПРАКТИЧНА РОБОТА № 7**

**«ШАБЛОНИ ФУНКЦІЙ ТА ПАРАМЕТРИЗОВАНІ ПІДПРОГРАМИ СОРТУВАННЯ»**

**Мета роботи:** Навчитися застосовувати різні шаблони функцій для вирішення задач при програмуванні мові С#.

using System;

namespace prac7\_console

{

class Car: ICloneable, IComparable

{

public int speed;

public string name;

public Car(int speed, string name)

{

this.speed = speed;

this.name = name;

}

public object Clone()

{

return new Car(speed, name);

}

public int CompareTo(object other)

{

Car otherCar = other as Car;

if (speed>otherCar.speed)

return 1;

else if(speed<otherCar.speed)

return -1;

else

return 0;

}

public static bool operator >(Car first, Car second)

{

return first.CompareTo(second)>0;

}

public static bool operator <(Car first, Car second)

{

return first.CompareTo(second) < 0;

}

public static bool operator >=(Car first, Car second)

{

return first.speed>=second.speed;

}

public static bool operator <=(Car first, Car second)

{

return first.speed <= second.speed;

}

public static bool operator ==(Car first, Car second)

{

return first.speed == second.speed;

}

public static bool operator !=(Car first, Car second)

{

return first.speed != second.speed;

}

public override string ToString()

{

return $"{name} - {speed}";

}

public override bool Equals(object obj)

{

return obj is Car car &&

speed == car.speed &&

name == car.name;

}

public override int GetHashCode()

{

return HashCode.Combine(speed, name);

}

}

internal class Program

{

public static void Sort<T>(T[] arr, bool direction = false) where T : IComparable

{

if (direction)

{

for (int i = 1; i < arr.Length; ++i)

{

T key = arr[i];

int j = i - 1;

while (j >= 0 && key.CompareTo(arr[j]) > 0)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

else

{

for (int i = 1; i < arr.Length; ++i)

{

T key = arr[i];

int j = i - 1;

while (j >= 0 && key.CompareTo(arr[j]) < 0)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

}

static void Main(string[] args)

{

int[] array = { 6, 8, 5, 4, 2, 9, 10, 8 };

Sort(array, true);

foreach (var item in array)

{

Console.WriteLine(item);

}

Console.WriteLine("Integer array\n");

double[] doubles = { 0.68, 8.3, 7.6, 3.8, 0.5 };

Sort(doubles);

foreach (var item in doubles)

{

Console.WriteLine(item);

}

Console.WriteLine("Double array\n");

Car[] cars = { new Car(120, "Volvo"), new Car(80, "Bugatti"), new Car(200, "Ferrari") };

Sort(cars);

foreach (var item in cars)

{

Console.WriteLine(item);

}

Console.WriteLine("Car array\n");

}

}

}

Клас Program має опис параметризованої підпрограми сортування public static void Sort<T> котра реалізовує інтерфейс Icomparable.

public static void Sort<T>(T[] arr, bool direction = false) where T : IComparable

{

if (direction)

{

for (int i = 1; i < arr.Length; ++i)

{

T key = arr[i];

int j = i - 1;

while (j >= 0 && key.CompareTo(arr[j]) > 0)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

else

{

for (int i = 1; i < arr.Length; ++i)

{

T key = arr[i];

int j = i - 1;

while (j >= 0 && key.CompareTo(arr[j]) < 0)

{

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

}

В цьому класі знаходиться статичний метод Main, що ініціалізовує масиви, та викликає підпрограму сортування передаючи туди дані масиви. Реалізований вивід відсортованих масивів у консоль.

static void Main(string[] args)

{

int[] array = { 6, 8, 5, 4, 2, 9, 10, 8 };

Sort(array, true);

foreach (var item in array)

{

Console.WriteLine(item);

}

Console.WriteLine("Integer array\n");

double[] doubles = { 0.68, 8.3, 7.6, 3.8, 0.5 };

Sort(doubles);

foreach (var item in doubles)

{

Console.WriteLine(item);

}

Console.WriteLine("Double array\n");

Car[] cars = { new Car(120, "Volvo"), new Car(80, "Bugatti"), new Car(200, "Ferrari") };

Sort(cars);

foreach (var item in cars)

{

Console.WriteLine(item);

}

Console.WriteLine("Car array\n");

}

Клас Car об’єктів масиву, призначений для сортування. Має перевантажені операції порівняння <, <=, ==, >=, >, =.

class Car: ICloneable, IComparable

{

public int speed;

public string name;

public Car(int speed, string name)

{

this.speed = speed;

this.name = name;

}

public object Clone()

{

return new Car(speed, name);

}

public int CompareTo(object other)

{

Car otherCar = other as Car;

if (speed>otherCar.speed)

return 1;

else if(speed<otherCar.speed)

return -1;

else

return 0;

}

public static bool operator >(Car first, Car second)

{

return first.CompareTo(second)>0;

}

public static bool operator <(Car first, Car second)

{

return first.CompareTo(second) < 0;

}

public static bool operator >=(Car first, Car second)

{

return first.speed>=second.speed;

}

public static bool operator <=(Car first, Car second)

{

return first.speed <= second.speed;

}

public static bool operator ==(Car first, Car second)

{

return first.speed == second.speed;

}

public static bool operator !=(Car first, Car second)

{

return first.speed != second.speed;

}

public override string ToString()

{

return $"{name} - {speed}";

}

public override bool Equals(object obj)

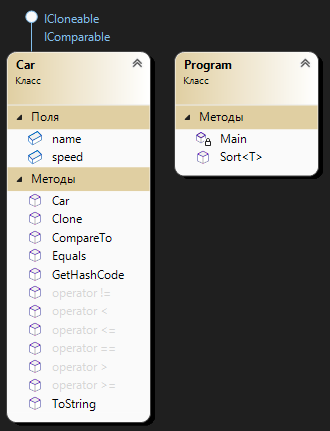
{

return obj is Car car &&

speed == car.speed &&

name == car.name;

}

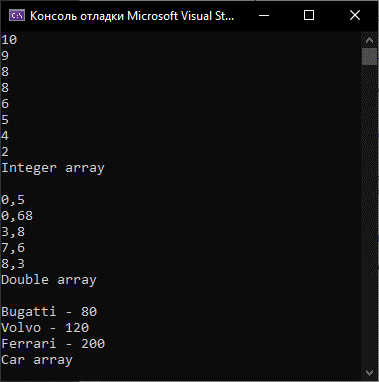
 public override int GetHashCode()

{

return HashCode.Combine(speed, name);

}

}



**ПРАКТИЧНА РОБОТА № 8**

**«ІНТЕРФЕЙСИ»**

**ВАРІАНТ №5**

**Мета**: Закріпити вміння реалізовувати класи, методи та конструктори. Навчитися створювати інтерфейси.

Ієрархія для завдання: Організація, страхова компанія, нафтогазова компанія, завод.

Повний лістинг коду:

using System;

using System.Collections.Generic;

namespace prac3\_console

{

public interface IInfo

{

public void GetInfo();

}

public interface IPayday

{

public void Pay();

}

public abstract class Organization

{

public string name;

public string industry;

public string area\_served;

public Organization(string name, string industry, string area\_served)

{

this.name = name;

this.industry = industry;

this.area\_served = area\_served;

}

}

public class Factory : Organization, IPayday, IInfo

{

public string head;

public int number\_of\_employees;

public string city;

public void GetInfo()

{

Console.WriteLine($"Factory: \n\tName of organisation: {name}");

Console.WriteLine("\tName: {0}\n\tHead: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

name, head, number\_of\_employees, city);

}

public void Pay()

{

Console.WriteLine($"{number\_of\_employees} employees was paid their salary");

}

public Factory(string name, string industry, string area\_served, string head, string city, int number\_of\_employees) : base(name, industry, area\_served)

{

this.head = head;

this.city = city;

this.number\_of\_employees = number\_of\_employees;

}

}

public class Insurance : Organization, IInfo

{

public int client\_number;

public string speciality;

public void GetInfo()

{

Console.WriteLine($"Insurance: \n\tName of organisation: {name}");

Console.WriteLine("\tNumber of clients: {0}\n\tSpecialization: {1}\n", client\_number, speciality);

}

public Insurance(string name, string industry, string area\_served, int client\_number, string speciality) : base(name, industry, area\_served)

{

this.client\_number = client\_number;

this.speciality = speciality;

}

}

public class BuildingCompany : Organization, IPayday, IInfo

{

public string speciality;

public string city;

public int number\_of\_employees;

public void GetInfo()

{

Console.WriteLine($"Building Company: \n\tName of organisation: {name}");

Console.WriteLine("\tName: {0}\n\tSpeciality: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

name, speciality, number\_of\_employees, city);

}

public void Pay()

{

Console.WriteLine($"{number\_of\_employees} employees was paid their salary");

}

public BuildingCompany(string name, string industry, string area\_wide, string speciality, string city, int number\_of\_employees) : base(name, industry, area\_wide)

{

this.speciality = speciality;

this.city = city;

this.number\_of\_employees = number\_of\_employees;

}

}

internal class Program

{

static void Main(string[] args)

{

Action action = new Action(() => Console.WriteLine("Delegate work started\n"));

Factory f = new Factory("Oil Co.", "oil", "Country", "Popolski", "Kyiv", 900);

BuildingCompany bc = new BuildingCompany("Building Co.", "skys", "City", "Houses", "Odesa", 1900);

Insurance ins = new Insurance("Insurance Co. ", "Money", "City", 300, "Cars");

action += f.Pay;

action += f.GetInfo;

action += bc.Pay;

action += bc.GetInfo;

action += ins.GetInfo;

action.Invoke();

}

}

}

Описано 2 інтерфейси IInfo та IPayday . У першому створений метод GetInfo() у другому Pay().

public interface IInfo

{

public void GetInfo();

}

public interface IPayday

{

public void Pay();

}

Клас Organization є абстрактним класом. В ньому описано декілька змінних для організацій та конструктор за цими змінними.

public abstract class Organization

{

public string name;

public string industry;

public string area\_served;

public Organization(string name, string industry, string area\_served)

{

this.name = name;

this.industry = industry;

this.area\_served = area\_served;

}

}

Клас Factory наслідує клас Organization та реалізовує інтерфейси IPayday, IInfo. Викликає обидва методи цих інтерфейсів та має конструктор базований на констрикторі основного класу.

public class Factory : Organization, IPayday, IInfo

{

public string head;

public int number\_of\_employees;

public string city;

public void GetInfo()

{

Console.WriteLine($"Factory: \n\tName of organisation: {name}");

Console.WriteLine("\tName: {0}\n\tHead: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

name, head, number\_of\_employees, city);

}

public void Pay()

{

Console.WriteLine($"{number\_of\_employees} employees was paid their salary");

}

public Factory(string name, string industry, string area\_served, string head, string city, int number\_of\_employees) : base(name, industry, area\_served)

{

this.head = head;

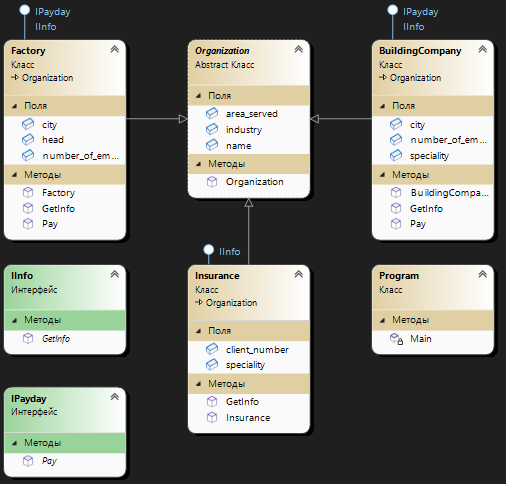
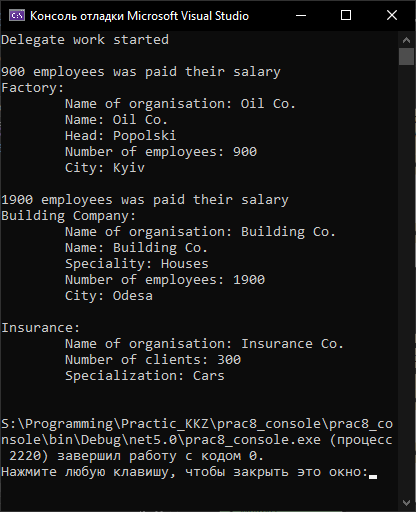
this.city = city;

this.number\_of\_employees = number\_of\_employees;

}

}

Класи BuildingCompany та Insurance мають таку ж реалізацію як і клас Factory.



**ПРАКТИЧНА РОБОТА № 9**

**«ПОДІЇ ТА ДЕЛЕГАТИ»**

**Мета**: Закріпити вміння при розробці класів забезпечити реакцію об’єктів на різні події та повідомлення.

using System;

namespace prac3\_console

{

public interface IInfo

{

public void GetInfo();

}

public interface IPayday

{

public void Pay();

}

public abstract class Organization

{

public string name;

public string industry;

public string area\_served;

public Organization(string name, string industry, string area\_served)

{

this.name = name;

this.industry = industry;

this.area\_served = area\_served;

}

}

public class Factory : Organization, IPayday, IInfo

{

public string head;

public int number\_of\_employees;

public string city;

public void GetInfo()

{

Console.WriteLine($"Factory: \n\tName of organisation: {name}");

Console.WriteLine("\tName: {0}\n\tHead: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

name, head, number\_of\_employees, city);

}

public void Pay()

{

Console.WriteLine($"{number\_of\_employees} employees was paid their salary");

}

public Factory(string name, string industry, string area\_served, string head, string city, int number\_of\_employees) : base(name, industry, area\_served)

{

this.head = head;

this.city = city;

this.number\_of\_employees = number\_of\_employees;

}

}

public class Insurance : Organization, IInfo

{

public int client\_number;

public string speciality;

public void GetInfo()

{

Console.WriteLine($"Insurance: \n\tName of organisation: {name}");

Console.WriteLine("\tNumber of clients: {0}\n\tSpecialization: {1}\n", client\_number, speciality);

}

public Insurance(string name, string industry, string area\_served, int client\_number, string speciality) : base(name, industry, area\_served)

{

this.client\_number = client\_number;

this.speciality = speciality;

}

}

public class BuildingCompany : Organization, IPayday, IInfo

{

public string speciality;

public string city;

public int number\_of\_employees;

public void GetInfo()

{

Console.WriteLine($"Building Company: \n\tName of organisation: {name}");

Console.WriteLine("\tName: {0}\n\tSpeciality: {1}\n\tNumber of employees: {2}\n\tCity: {3}\n",

name, speciality, number\_of\_employees, city);

}

public void Pay()

{

Console.WriteLine($"{number\_of\_employees} employees was paid their salary");

}

public BuildingCompany(string name, string industry, string area\_wide, string speciality, string city, int number\_of\_employees) : base(name, industry, area\_wide)

{

this.speciality = speciality;

this.city = city;

this.number\_of\_employees = number\_of\_employees;

}

}

public class EventStore

{

public delegate Exception EventHandler();

public event EventHandler Notify1;

public event EventHandler Notify2;

public event EventHandler Notify3;

public event EventHandler Notify4;

public event EventHandler Notify5;

public event EventHandler Notify6;

public event EventHandler Notify7;

public Exception StackOverflowException()

{

return Notify1.Invoke();

}

public Exception ArrayTypeMismatchException()

{

return Notify2.Invoke();

}

public Exception DivideByZeroException()

{

return Notify3.Invoke();

}

public Exception IndexOutOfRangeException()

{

return Notify4.Invoke();

}

public Exception InvalidCastException()

{

return Notify5.Invoke();

}

public Exception OutOfMemoryException()

{

return Notify6.Invoke();

}

public Exception OverflowException()

{

return Notify7.Invoke();

}

private EventStore SetEvents(EventStore es)

{

es.Notify1 += () => throw new StackOverflowException();

es.Notify2 += () => throw new ArrayTypeMismatchException();

es.Notify3 += () => throw new DivideByZeroException();

es.Notify4 += () => throw new IndexOutOfRangeException();

es.Notify5 += () => throw new InvalidCastException();

es.Notify6 += () => throw new OutOfMemoryException();

es.Notify7 += () => throw new OverflowException();

return es;

}

public EventStore()

{

SetEvents(this);

}

}

internal class Program

{

static void Main(string[] args)

{

Action action = new Action(() => Console.WriteLine("Delegate work started\n"));

Factory f = new Factory("agrr", "oil", "Country", "Popolski", "Kyiv", 900);

BuildingCompany bc = new BuildingCompany("Building Co.", "skys", "City", "Gladkevich", "Odesa", 1900);

Insurance ins = new Insurance("Insurance Co. ", "Money", "City", 300, "Cars");

action += f.Pay;

action += f.GetInfo;

action += bc.Pay;

action += bc.GetInfo;

action += ins.GetInfo;

action.Invoke();

EventStore es = new EventStore();

try

{

f.number\_of\_employees /= 0;

}

catch (Exception e)

{

es.DivideByZeroException();

}

}

}

}

До програми створеної у Практичній роботі №8 були додані наступні зміни:

Створений клас EventStore для збереження всіх подій помилок, їх перевизначення та підписання на події анонімних методів, що викликають помилки.

public class EventStore

{

public delegate Exception EventHandler();

public event EventHandler Notify1;

public event EventHandler Notify2;

public event EventHandler Notify3;

public event EventHandler Notify4;

public event EventHandler Notify5;

public event EventHandler Notify6;

public event EventHandler Notify7;

public Exception StackOverflowException()

{

return Notify1.Invoke();

}

public Exception ArrayTypeMismatchException()

{

return Notify2.Invoke();

}

public Exception DivideByZeroException()

{

return Notify3.Invoke();

}

public Exception IndexOutOfRangeException()

{

return Notify4.Invoke();

}

public Exception InvalidCastException()

{

return Notify5.Invoke();

}

public Exception OutOfMemoryException()

{

return Notify6.Invoke();

}

public Exception OverflowException()

{

return Notify7.Invoke();

}

private EventStore SetEvents(EventStore es)

{

es.Notify1 += () => throw new StackOverflowException();

es.Notify2 += () => throw new ArrayTypeMismatchException();

es.Notify3 += () => throw new DivideByZeroException();

es.Notify4 += () => throw new IndexOutOfRangeException();

es.Notify5 += () => throw new InvalidCastException();

es.Notify6 += () => throw new OutOfMemoryException();

es.Notify7 += () => throw new OverflowException();

return es;

}

public EventStore()

{

SetEvents(this);

}

}

**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)** \_\_\_Філіпенко Андрій Дмитрович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**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 | 13.03.20 | Ознайомлення з технікою безпеки і задачами практики |  |
| 2 | 14.03.20 –16.03.20 | Розробка алгоритму і написання коду програм до практичної роботи № 1 |  |
| 3 | 20.03.20 | Опрацювання літератури |  |
| 3 | 20.03.20 –22.03.20 | Розробка алгоритму і написання коду програми до практичної роботи № 2 |  |
| 4 | 23.03.20–24.03.20 | Розробка алгоритму і написання коду для практичної роботи № 3 |  |
| 5 | 27.03.20–29.03.20 | Розробка алгоритму і написання коду програми до практичної роботи № 4 |  |
| 6 | 30.03.20–31.03.20 | Розробка алгоритму і написання коду програми до практичної роботи № 5 |  |
| 7 | 03.03.20 | Розробка алгоритму і написання коду програми до практичної роботи № 6 |  |
| 8 | 04.03.20 | Розробка алгоритму і написання коду програми до практичної роботи № 8 |  |
| 9 | 05.03.20 | Розробка алгоритму і написання коду програми до практичної роботи № 9 |  |
| 10 | 11.04.20–13.03.20 | Оформлення звіту |  |
| 11 | 14.04.20 | 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)** Філіпенка Андрія Дмитровича\_\_\_\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | 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**

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

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