**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*:

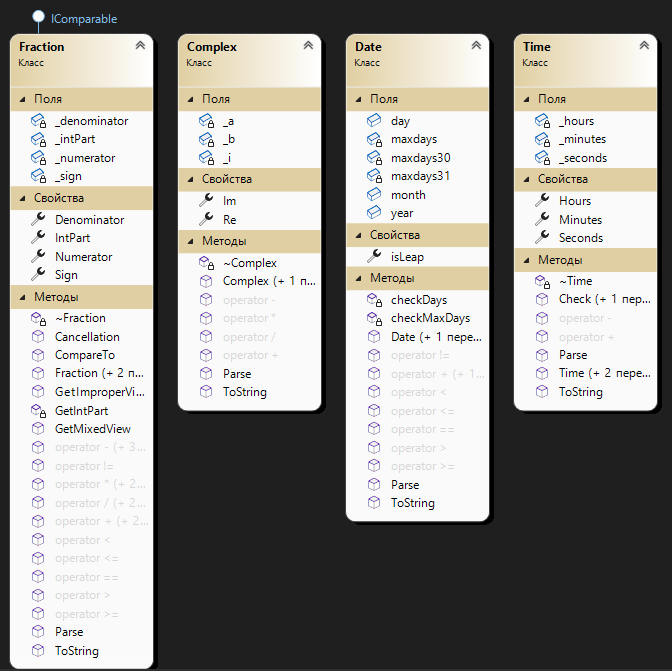
~Time()//деструктор

{

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)

{

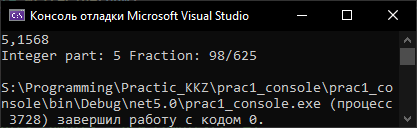
Fraction fraction1 = new Fraction(5.1568);

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);

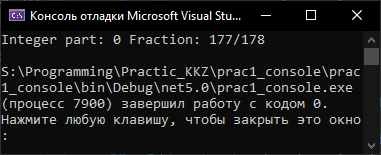
Fraction fraction2 = new Fraction(44, 89);

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);

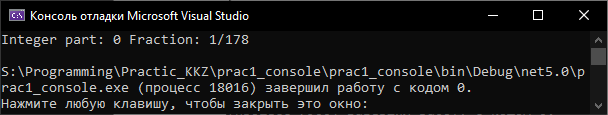
Fraction fraction2 = new Fraction(44, 89);

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);

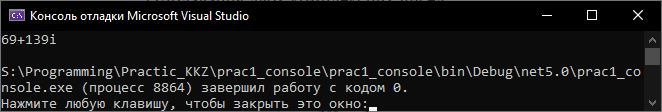
Complex complex2 = new Complex(44, 89);

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);

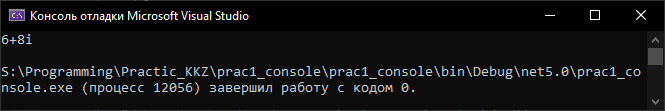
Complex complex2 = new Complex(4, 16);

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);

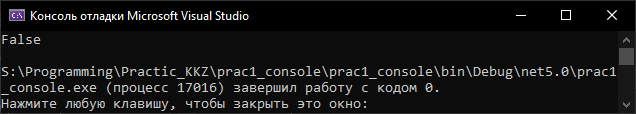
Date date2 = new Date(12, 12, 2005);

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);

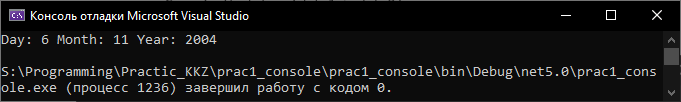
int days = 346;

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)

{

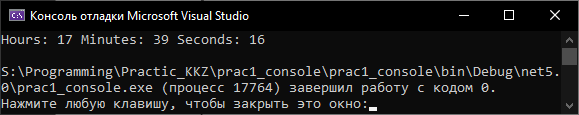
Time time1 = new Time(16, 52, 16);

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);

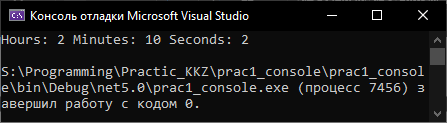
Time time2 = new Time(78, 121, 17);

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");

}

}

}

**Висновок:** в ході виконанння практичної роботи ми засвоїли основи створення класів в мові C#, набули практичних навичок застосування принципів поліморфізму об’єктно-орієнтованого програмування.