|  |
| --- |
| *Варіант 16.* |
| 1. Описати клас, який містять вказані поля і методи.   |  |  | | --- | --- | | **Клас** “комплексне число” – TComplex | | | **поля** | * для зберігання дійсної і уявної частин; | | **методи** | * конструктор без параметрів, конструктор з параметрами, конструктор копіювання; * введення/виведення даних; * перевантаження операторів +, –, \*, / . | |
| 2. Створити клас-нащадок TMComplex (комплексне число на площині) на основі класу TComplex. Додати методи визначення квадранта, у який попадає комплексне число, метод визначення відстані до початку координат. |
| 3. Створити програму-клієнт для тестування. |

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace my\_project

{

internal class Program

{

static void Main(string[] args)

{

TComplex a = new TComplex(10.5);

TComplex b = new TComplex();

TComplex c = new TComplex(b);

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

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

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

Console.WriteLine();

Console.WriteLine($"Сума={a + b}");

Console.WriteLine($"Різниця={a - b}");

Console.WriteLine($"Добуток={a \* b}");

Console.WriteLine($"Частка={a / b}");

Console.WriteLine();

var d = a / b;

Console.WriteLine(c + d);

Console.WriteLine(c - d);

Console.WriteLine(c \* d);

Console.WriteLine(c / d);

Console.WriteLine();

TMComplex tmc = new TMComplex(-10.4, -3);

Console.WriteLine(tmc);

Console.WriteLine();

Console.WriteLine(tmc.Quadrant());

Console.WriteLine(tmc.Distance());

Console.ReadLine();

}

}

public class TComplex

{

public double Number { get; set; }

public char Complex { get { return \_complex; } set { if (value == 'i' || value == ' ') \_complex = value; } } private char \_complex = '\_';

public TComplex(double n = 1, char i = 'i')

{

Number = n;

Complex = i;

}

public TComplex()

{

Console.Write("Введіть число: ");

Number = Convert.ToDouble(Console.ReadLine());

Complex = 'i';

}

public TComplex(TComplex a)

{

Number = a.Number;

Complex = a.Complex;

}

public override string ToString()

{

if (Complex == '\_') return " ";

else if (Complex == ' ' || Number == 0) return $"{Number:f1}";

else if (Number == 1) return $"{Complex}";

else if (Number == -1) return $"-{Complex}";

else return $"{Number:f1}{Complex}";

}

public static TComplex operator +(TComplex a, TComplex b)

{

if (a.Complex == ' ' && b.Complex == ' ')

return new TComplex(a.Number + b.Number, ' ');

else if (a.Complex == 'i' && b.Complex == 'i')

return new TComplex(a.Number + b.Number);

else

{

if (b.Number > 0)

Console.Write($"{a} + {b}");

else

{

b.Number \*= -1;

Console.Write($"{a} - {b}");

b.Number \*= -1;

}

return new TComplex(0, '\_');

}

}

public static TComplex operator -(TComplex a, TComplex b)

{

if (a.Complex == ' ' && b.Complex == ' ')

return new TComplex(a.Number - b.Number, ' ');

else if (a.Complex == 'i' && b.Complex == 'i')

return new TComplex(a.Number - b.Number);

else

{

if (b.Number > 0)

Console.Write($"{a} - {b}");

else

{

b.Number \*= -1;

Console.Write($"{a} + {b}");

b.Number \*= -1;

}

return new TComplex(0, '\_');

}

}

public static TComplex operator \*(TComplex a, TComplex b)

{

if (a.Complex == 'i' && b.Complex == 'i')

return new TComplex(-1 \* a.Number \* b.Number, ' ');

else if (a.Complex == ' ' && b.Complex == ' ')

return new TComplex(a.Number \* b.Number, ' ');

else

return new TComplex(a.Number \* b.Number);

}

public static TComplex operator /(TComplex a, TComplex b)

{

if (a.Complex == 'i' && b.Complex == 'i' || a.Complex == ' ' && b.Complex == ' ')

return new TComplex(a.Number / b.Number, ' ');

else

return new TComplex(a.Number / b.Number);

}

}

public class TMComplex : TComplex

{

public double X { get; set; }

public TMComplex(double x = 1, double n = 1, char i = 'i'):base(n, i)

{

X = x;

}

public override string ToString()

{

return ( Number >= 0 ) ? $"Z = {X} + {base.ToString()}" : $"Z = {X} + ({base.ToString()})";

}

public string Quadrant()

{

if (X > 0 && Number > 0) return "I";

else if (X > 0 && Number < 0) return "II";

else if (X < 0 && Number < 0) return "III";

else if (X < 0 && Number > 0) return "IV";

else return "0";

}

public double Distance()

{

return Complex == 'i' ? Math.Sqrt(Math.Pow(X, 2) - Math.Pow(Number, 2)) : Math.Sqrt(Math.Pow(X, 2) + Math.Pow(Number, 2));

}

}

}