Jose Cabrera – 301212338

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Complex

{

internal class Program

{

public class Complex

{

//Atributes

public int Real { get; }

public int Imaginary { get; }

public double Argument;

public double Modulus;

public static Complex Zero;

//Constructor

public Complex(int real = 0, int imaginary = 0)

{

Real = real;

Imaginary = imaginary;

Modulus = Math.Pow((Real \* Real) + (Imaginary \* Imaginary), 0.5);

Argument = Math.Tanh(Imaginary / Real);

}

//Methods

public override string ToString()

{

return $"({Real},{Imaginary})";

}

public static Complex operator +(Complex lhs, Complex rhs)

{

int real = lhs.Real + rhs.Real;

int imaginary = lhs.Imaginary + rhs.Imaginary;

return new Complex(real, imaginary);

}

public static Complex operator -(Complex lhs, Complex rhs)

{

int real = lhs.Real - rhs.Real;

int imaginary = lhs.Imaginary - rhs.Imaginary;

return new Complex(real, imaginary);

}

public static bool operator ==(Complex lhs, Complex rhs)

{

bool result;

if (lhs.Real == rhs.Real && lhs.Imaginary == rhs.Imaginary)

{

result = true;

}

else

{

result = false;

}

return result;

}

public static bool operator !=(Complex lhs, Complex rhs)

{

bool result;

if (lhs.Real == rhs.Real && lhs.Imaginary == rhs.Imaginary)

{

result = false;

}

else

{

result = true;

}

return result;

}

}

static void Main(string[] args)

{

Complex c0 = new Complex(-2, 3);

Complex c1 = new Complex(-2, 3);

Complex c2 = new Complex(1, -2);

Console.WriteLine($"{c0}");

Console.WriteLine(c1);

Console.WriteLine(c2);

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

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

Complex c3 = c1 + c2;

Console.WriteLine($"{c3} in polar form is {c3.Modulus:f2}cis({c3.Argument:f2})");

Console.WriteLine($"{c0} {(c0 == c1 ? "=" : "!=")} {c1}");

Console.WriteLine($"{c0} {(c0 == c2 ? "=" : "!=")} {c2}");

}

}

}

This exercise you will be exploring more about properties and be introduced to overloading operators.

# The Complex class

We are going to model a type that will behave like a complex number in higher mathematics. We will try to reduce the complexity of this type by not implementing all of the normal behaviors. There are 11 members as shown in the class diagram below.

|  |
| --- |
| **Complex**  Class |
| **Properties**  + «property setter absent» Real : **int**  + «property setter absent» Imaginary : **int**  + «computed property» Argument : **double**  + «computed property» Modulus : **double**  $+ «factory property» Zero : Complex |
| **Methods**  + «constructor» Complex(  real = 0 : **int**,  imaginary = 0 : **int**)  + ToString() : **string**  $+ «operator» + (lhs : Complex, rhs : Complex) : Complex  $+ «operator» - (lhs : Complex, rhs : Complex) : Complex  $+ «operator» == (lhs : Complex, rhs : Complex) : **bool**  $+ «operator» != (lhs : Complex, rhs : Complex) : **bool** |

#### Description of class members

##### Fields:

There are no fields in this type

##### Properties:

The first two properties (**Real** and **Imaginary**) are auto-implemented and the rest (**Modulus, Argument** and **Zero**) are not. These properties are calculated on the fly i.e. whenever they are required they are calculated.

Auto-implemented property is what we have been using so far. You may not attach code in either the getter or the setter parts.

**Real** – this int represents the real part of this type. This is an auto-implemented property, the getter is public and the setter is **absent**.

**Imaginary** – this int represents the imaginary part of this type. This is an auto-implemented property, the getter is public and the setter is **absent**.

**Modulus** – this double represents the complex modulus or length of this object. It is calculated as the square root of the sum of the square of the Real and Imaginary parts of this type []. The getter is public and the setter is **absent**.

Both Modulus and Argument are computed property and Zero is a factory property.

**Argument** – this double represents the complex argument or the angle it make with the horizontal axis. It is calculated as the inverse tan of the ratio of the Real to the Imaginary part of this type []. The getter is public and the setter is **absent**.

**Zero** – this static property returns a new complex object with both the real and the imaginary parts equal to 0. The getter is public and of course there is no setter.

##### Constructor:

**Complex(int real, int imaginary)** – This is constructor assigns the arguments to the appropriate field

##### Methods

**ToString()** – This is a public method overrides the corresponding method in the object class to return a stringify form of the object. For this you return the Real and the Imaginary properties as an ordered pair.

##### Operators:

public static Complex operator +(Complex lhs, Complex rhs)

The return type

The operator that you want to overload

First operand

Second operand

Signals that you are overloading an operator

+ – You will overload the plus operator to add the two numbers. Copy the code below into your type declaration to overload the + operator.

public static Complex operator +(Complex lhs, Complex rhs)

{

int real = lhs.Real + rhs.Real;

int imaginary = lhs.Imaginary + rhs.Imaginary;

return new Complex(real, imaginary);

}

**-**  – You will also overload the minus operator. Examine the code above and then try to implement this operator.

Some operators have to be overloaded in pairs. This is one of them.

So you will also have to overload the **==** and the **!=** operator at the same time

**==**  – You will also overload the equal-equal operator. Examine the code above and then try to implement this operator. What should the return type of the method be?

Test Harness

Insert the following code statements in your Program.cs file:

Complex c0 = new Complex(-2, 3);

Complex c1 = new Complex(-2, 3);

Complex c2 = new Complex(1, -2);

Console.WriteLine($"{c0}");

Console.WriteLine(c1);

Console.WriteLine(c2);

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

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

Complex c3 = c1 + c2;

Console.WriteLine($"{c3} in polar form is {c3.Modulus:f2}cis({c3.Argument:f2})");

Console.WriteLine($"{c0} {(c0 == c1 ? "=" : "!=")} {c1}");

Console.WriteLine($"{c0} {(c0 == c2 ? "=" : "!=")} {c2}");

### Additional tasks

1. Try to overload the \* operator. Multiplication of two complex numbers is defined by the following relation: <a, b> \* <c, d> = <ac-bd, ad+bc>.

Insert the proper code statements in your main to show the operation of this operator.

1. Try to overload the unary - operator. This operator simply changes the sign of the operand: -<a, b> = <-a, -b>.

Again, insert the proper code statements in your main to show the operation of this operator.