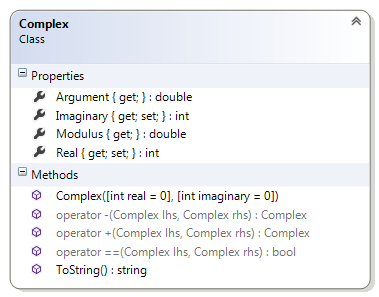
This exercise you will be exploring more about properties and be introduced to overloading operators.

# 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 9 members as shown in the class diagram below.

#### Description of class members

##### Fields:

There are no fields in this type

##### Properties:

The second and fourth properties (**Real** and **Imaginary**) are auto-implemented and the first and third (**Modulus** and **Argument**) are not. The latter two 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. A backing field is not required and both set and get must be present. 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 private.

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

**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.

**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.

##### 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:

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

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.

**==**  – 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?

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

### 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);

Complex c3 = c1 + c2;

Console.WriteLine("{0} + {1} = {2}", c1, c2, c3);

Console.WriteLine("{0} - {1} = {2}", c1, c2, c1 - c2);

Console.WriteLine("{0} in polar form is {1:f2}cis({2:f2})", c3, c3.Modulus, c3.Argument);

Console.WriteLine("{0} {1} {2}", c0, (c0 == c1) ? "=" : "!=", c1);

Console.WriteLine("{0} {1} {2}", c0, (c0 == c2) ? "=" : "!=", c2);

### Additional tasks

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.