Assignment 4: Operator Overloading and Unit Testing

Complex Number Implementation Report

1. Overview

This assignment required implementing a Complex class that represents complex numbers (numbers with real and imaginary parts) and overloading all necessary arithmetic and comparison operators. Additionally, comprehensive unit tests were created to validate all operator functionality.

A complex number is represented as: **a + bi**, where:

- **a** = Real part
- **b** = Imaginary part
- **i** = Imaginary unit ($i^2 = -1$)

Complex Number Rules:

- Addition: $(a + bi) + (c + di) = (a+c) + (b+d)i$
- Subtraction: $(a + bi) - (c + di) = (a-c) + (b-d)i$
- Multiplication: $(a + bi)(c + di) = (ac-bd) + (ad+bc)i$
- Division: $(a + bi)/(c + di) = [(ac+bd) + (bc-ad)i] / (c^2 + d^2)$
- $i^2 = -1$ (fundamental property)

2. Complex Class Structure

Properties

The Complex class contains two properties:

```
public double Real { get; set; }    // Stores the real part

public double Imag { get; set; }    // Stores the imaginary part
```

ToString() Method

Formats the complex number for display:

- If imaginary part is positive: 3 + i4
- If imaginary part is negative: 3 - i2

3. Implemented Operators

Addition Operator (+)

Purpose: Adds two complex numbers

Formula: (a + bi) + (c + di) = (a + c) + (b + d)i

Example: (3 + 4i) + (2 + 1i) = (5 + 5i)

Test Cases:

- Positive numbers: 3+4i + 2+1i = 5+5i
- Negative numbers: -3-4i + -2-1i = -5-5i
- Mixed numbers: 5-2i + -3+6i = 2+4i
- With zero: 3+4i + 0+0i = 3+4i

Subtraction Operator (-)

Purpose: Subtracts one complex number from another

Formula: (a + bi) - (c + di) = (a - c) + (b - d)i

Example: (5 + 6i) - (2 + 3i) = (3 + 3i)

Test Cases:

- Positive numbers: 5+6i - 2+3i = 3+3i
- Negative numbers: -5-6i - -2-3i = -3-3i
- Mixed numbers: 7-4i - -2+3i = 9-7i
- With zero: 4+5i - 0+0i = 4+5i

Multiplication Operator (*)

Purpose: Multiplies two complex numbers

Formula: (a + bi)(c + di) = (ac - bd) + (ad + bc)i

Example: (2 + 3i)(4 + 5i) = (8 - 15) + (10 + 12)i = -7 + 22i

Explanation:

- Real part: (2 × 4) - (3 × 5) = 8 - 15 = -7
- Imaginary part: (2 × 5) + (3 × 4) = 10 + 12 = 22

Test Cases:

- Positive numbers: 2+3i × 4+5i = -7+22i
- Negative numbers: -2-3i × -4-5i = -7+22i
- Mixed numbers: 3-2i × -1+4i = 5+14i

- With zero: 5+3i × 0+0i = 0+0i

Division Operator (/)

Purpose: Divides one complex number by another

Formula: $(a + bi)/(c + di) = [(ac + bd) + (bc - ad)i] / (c^2 + d^2)$

Example: (10 + 5i)/(2 + 1i) = 5 + 0i

Explanation:

- Multiply numerator and denominator by conjugate of denominator
- $(10 + 5i)(2 - 1i) / (2^2 + 1^2) = (20 + 5) / 5 = 5$

Test Cases:

- Positive numbers: 10+5i / 2+1i = 5+0i
- Negative numbers: -8-4i / -2-1i = 4+0i
- Result with imaginary: 3+4i / 1+2i = 2.2-0.4i
- Division by zero: Throws DivideByZeroException

Equality Operator (==)

Purpose: Compares two complex numbers for equality

Logic: Both real and imaginary parts must be equal

Example: (3 + 4i) == (3 + 4i) returns true

Test Cases:

- Identical numbers: 3+4i == 3+4i -> true
- Different numbers: 3+4i == 2+5i -> false
- Both zero: 0+0i == 0+0i -> true
- Negative numbers: -5-7i == -5-7i -> true

Inequality Operator (!=)

Purpose: Compares two complex numbers for inequality

Logic: Returns true if either real or imaginary parts differ

Example: (3 + 4i) != (2 + 5i) returns true

Test Cases:

- Different numbers: 3+4i != 2+5i -> true
- Identical numbers: 3+4i != 3+4i -> false

- Different real parts: 5+4i != 3+4i -> true
- Different imaginary parts: 3+5i != 3+4i -> true

Increment Operator (++)

Purpose: Increments the real part by 1

Logic: c.Real++

Example: 3+4i after ++ becomes 4+4i

Test Cases:

- Positive number: 3+4i -> 4+4i
- Negative number: -2-5i -> -1-5i
- Zero: 0+0i -> 1+0i
- Only imaginary: 0+5i -> 1+5i

Decrement Operator (--)

Purpose: Decrements the real part by 1

Logic: c.Real--

Example: 5+3i after -- becomes 4+3i

Test Cases:

- Positive number: 5+3i -> 4+3i
- Negative number: -3-5i -> -4-5i
- Zero: 0+0i -> -1+0i
- Only imaginary: 0+7i -> -1+7i

4. Unit Testing Overview

Testing Framework

- **Framework:** MSTest
- **Total Tests:** 32
- **Test Results:** 32 Passed, 0 Failed, 0 Skipped

Test Structure

Each test follows this pattern:

1. **Arrange:** Set up test data

2. **Act:** Perform the operation
3. **Assert:** Verify the results

Example Test

```
[TestMethod]
public void AdditionTest_PositiveNumbers()
{
   // Arrange
   Complex c1 = new Complex { Real = 3, Imag = 4 };
   Complex c2 = new Complex { Real = 2, Imag = 1 };
   Complex expected = new Complex { Real = 5, Imag = 5 };

   // Act
   Complex result = c1 + c2;

   // Assert
   Assert.AreEqual(expected.Real, result.Real);
   Assert.AreEqual(expected.Imag, result.Imag);
}
```

5. Test Results Summary

Operator	**Test Category**	**Count**	**Status**
Addition	Positive, Negative, Mixed, Zero	4	Pass
Subtraction	Positive, Negative, Mixed, Zero	4	Pass
Multiplication	Positive, Negative, Mixed, Zero	4	Pass
Division	Positive, Negative, Mixed, Exception	4	Pass
Equality	Identical, Different, Zero, Negative	4	Pass
Inequality	Different, Identical, Different Real, Different Imag	4	Pass
Increment	Positive, Negative, Zero, Only Imag	4	Pass
Decrement	Positive, Negative, Zero, Only Imag	4	Pass
TOTAL		32	**All Pass**

6. Key Implementation Details

Exception Handling

The division operator includes proper exception handling:

```
if (rhs.Real == 0 && rhs.Imag == 0)
    throw new DivideByZeroException("Cannot divide by zero complex number");
```

Floating Point Precision

Division tests use a tolerance value (0.0001) for comparing results:

```
Assert.AreEqual(expected.Real, result.Real, 0.0001);
```

Method Overrides

Additional overrides for proper object behavior:

- Equals(object obj) - Custom equality comparison
- GetHashCode() - Hash code generation for collections
- ToString() - String representation