# COMP2500 실습 2

## 목차



본 과제는 컴퓨터에서 해야 하는 과제입니다. 코드 작성이 끝났다면 실습 1에서 만들었던 깃 저장소에 커밋 및 푸시를 하고 슬랙을 통해 자동으로 채점을 받으세요.

대부분의 현대 프로그래밍 언어는 자체적으로 여러 숫자 자료형과 수학 연산을 지원합니다. Java에는 byte, int, double 등의 숫자 자료형이 있고 복잡한 수학 연산을 위해 `java.lang.Math` 클래스도 있죠. 하지만 복소수(complex number)를 표현하는 방법은 빠져있습니다. 이제 클래스가 뭔지도 배워봤으니 이 실습에서는 복소수를 직접 모델링해보겠습니다.

복소수는 `a + bi`의 형태로 표현하는 숫자로 `a`는 실수부(real part), `b`는 허수부(imaginary part)를 나타냅니다. `i`는 허수 단위(imaginary unit)로 그 값은 $\sqrt{-1}$입니다. 복소수는 $x^2 = -1$처럼 해가 없는 방정식을 표현할 수 있게 해 주며, 특히 신호처리나 전자기학 같은 다양한 물리학 분야에서 사용합니다.

혹시라도 나중에 물리 관련 프로그래밍을 하게 된다면 복소수 클래스를 만들어 사용하실 가능성이 큽니다. ^^

# 1. 프로젝트를 준비한다

1. 실습 1을 반드시 끝내도록 하세요. 그래야만 본인의 컴퓨터에 `Lab2` 폴더가 올바르게 설정되어 있을 겁니다.
2. `Lab2` 폴더로 이동합니다.
3. `src/academy/pocu/comp2500/lab2` 폴더에 `ComplexNumber.java` 파일을 추가합니다.
4. 다음 내용을 위 파일에 추가합니다.

```
package academy.pocu.comp2500.lab2;

public class ComplexNumber {

}
```

# 2. `ComplexNumber` 구현하기

## 전반적인 규칙

- 어떤 클래스 대신 사용할 수 있는 기본 자료형이 존재한다면(예: Boolean/boolean, Integer/int) 그걸 대신 사용하세요. 기본 자료형 대신 클래스 버전이 허용되는 경우는 제네릭(generic) 클래스의 타입(type) 매개변수로 사용할 때 뿐입니다.
- 이 실습에서 제공하는 테스트 코드를 사용하려면 `-ea` VM 옵션으로 `assert` 키워드를 활성화시켜야 합니다. 본인 코드에서 `assert`를 사용한다면 이 옵션을 켜고 프로그램을 실행하세요.
- ComplexNumber를 변경 불가능한 클래스처럼 다뤄주세요. 즉, 일단 개체를 생성한 뒤에는 그 상태를 바꿀 수 없습니다.
- 다음은 복소수를 다룰 때 알면 좋은 몇 가지 용어입니다.
    - 순실수(purely real): 복소수에 허수부가 없다면(허수부가 0) 그 복소수를 순실수라고 부릅니다.

  - 순허수(purely imaginary): 실수부가 없는(실수부가 0) 복소수를 순허수라고 부릅니다.
  - 켤레(conjugate) 복소수: `a + bi`란 복소수의 켤레 복소수는 `a - bi`입니다.
- 다음은 복소수의 연산 규칙입니다.
  - 덧셈: 실수부는 실수부끼리 허수부는 허수부끼리 따로 더합니다.

```
(a + bi) + (c + di) = (a + c) + (b + d)i
```

  - 뺄셈: 실수부는 실수부끼리 허수부는 허수부끼리 따로 뺍니다.

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

  - 곱셈: 두 복소수를 분배 법칙, 결합 법칙에 따라 곱합니다.

```
(a + bi)(c + di) = ac + adi + bci + bdi^2   // i^2 = -1 임
                 = (ac - bd) + (ad + bc)i
```

  - 나눗셈: 나눗셈은 켤레 복소수와 곱셈을 이용해 수행합니다.

```
(a + bi) / (c + di) = [(a + bi) / (c + di)] * [(c - di) / (c - di)]
                    = (a + bi)(c - di) / (c^2 + d^2)
```

## 2.1 생성자를 구현하고 멤버 변수를 추가한다

- 이 클래스에는 2개의 멤버 변수가 있어야 합니다.
  - 복소수의 실수부: `double real`
  - 복소수의 허수부: `double imaginary`
- 이 클래스에는 3개의 생성자가 있어야 합니다.
  - 매개변수 없는 생성자: `real`과 `imaginary`를 0.0으로 초기화합니다.
  - 매개변수가 `double real` 뿐인 생성자: `imaginary`를 0.0으로 초기화합니다.
  - 매개변수가 `double real, double imaginary`인 생성자

```
ComplexNumber num1 = new ComplexNumber();
ComplexNumber num2 = new ComplexNumber(0.05);
ComplexNumber num3 = new ComplexNumber(1.5, -2.0);
```

## 2.2 `isReal()` 메서드를 구현한다

- `isReal()` 메서드는 인자를 받지 않습니다.
- 복소수가 순실수인 경우 `true`를, 아닌 경우 `false`를 반환합니다.

```
ComplexNumber num1 = new ComplexNumber(2.1, -1.1);
ComplexNumber num2 = new ComplexNumber(1.2);

boolean isReal = num1.isReal(); // isReal: false
isReal = num2.isReal();         // isReal: true
```

## 2.3 `isImaginary()` 메서드를 구현한다

- `isImaginary()` 메서드는 인자를 받지 않습니다.
- 복소수가 순허수인 경우 `true`를, 아닌 경우 `false`를 반환합니다.

```
ComplexNumber num1 = new ComplexNumber(2.1, -1.1);
ComplexNumber num2 = new ComplexNumber(0.0, 2.5);

boolean isImaginary = num1.isImaginary(); // isImaginary: false
isImaginary = num2.isImaginary();         // isImaginary: true
```

## 2.4 `getConjugate()` 메서드를 구현한다

- `getConjugate()` 메서드는 인자를 받지 않습니다.
- 현재 복소수의 켤레 복소수를 반환합니다.

```
ComplexNumber num1 = new ComplexNumber(2.1, -1.1);

ComplexNumber conjugate = num1.getConjugate(); // conjugate: 2.1 + 1.1i
```

## 2.5 add() 메서드를 구현한다

- add() 메서드는 다음의 인자를 받습니다.
  - 더할 복소수: ComplexNumber num
- 현재 인스턴스와 num 을 합한 결과를 반환합니다.

```
ComplexNumber num1 = new ComplexNumber(0.5, -1.5);
ComplexNumber num2 = new ComplexNumber(-2.5, 4.0);

ComplexNumber sum = num1.add(num2); // sum: -2.0 + 2.5i
```

## 2.6 subtract() 메서드를 구현한다

- subtract() 메서드는 다음의 인자를 받습니다.
  - 뺄 복소수: ComplexNumber num
- 현재 인스턴스에서 num 을 뺀 결과를 반환합니다.

```
ComplexNumber num1 = new ComplexNumber(0.5, -1.5);
ComplexNumber num2 = new ComplexNumber(-2.5, 4.0);

ComplexNumber diff = num1.subtract(num2); // diff: 3.0 -5.5i
```

## 2.7 multiply() 메서드를 구현한다

- multiply() 메서드는 다음의 인자를 받습니다.
  - 곱할 복소수: ComplexNumber num
- 현재 인스턴스와 num 의 곱을 반환합니다.

```
ComplexNumber num1 = new ComplexNumber(0.5, -1.5);
ComplexNumber num2 = new ComplexNumber(-2.5, 4.0);

ComplexNumber product = num1.multiply(num2); // product: 4.75 + 5.75i
```

## 2.8 divide() 메서드를 구현한다

- divide() 메서드는 다음의 인자를 받습니다.
  - 나눗셈에 분모로 사용할 복소수: ComplexNumber num
- 현재 인스턴스를 num 으로 나눈 결과를 반환합니다.

```
ComplexNumber num1 = new ComplexNumber(0.5, -1.5);
ComplexNumber num2 = new ComplexNumber(-2.5, 4.0);

ComplexNumber quotient = num1.divide(num2); // quotient: -0.32584 + 0.07865i
```

# 3. 본인 컴퓨터에서 테스트하는 법

- Program.java 를 아래처럼 바꾼 뒤 실행하세요.

```java
package academy.pocu.comp2500.lab2.app;

import academy.pocu.comp2500.lab2.ComplexNumber;

public class Program {
    private static final double DOUBLE_EPSILON = 0.00001;

    public static void main(String[] args) {
        ComplexNumber num1 = new ComplexNumber();
        ComplexNumber num2 = new ComplexNumber(10.0);
        ComplexNumber num3 = new ComplexNumber(2.5, -5.1);
        ComplexNumber num4 = new ComplexNumber(0.0, 0.0012);

        assert (num1.real == 0.0);
        assert (num1.imaginary == 0.0);

        assert (num2.real == 10.0);
        assert (num2.imaginary == 0.0);

        assert (num3.real == 2.5);
        assert (num3.imaginary == -5.1);

        assert num2.isReal();
        assert !num3.isReal();

        assert num4.isImaginary();
        assert !num3.isImaginary();

        ComplexNumber num3Conjugate = num3.getConjugate();

        assert num3Conjugate.real == 2.5;
        assert num3Conjugate.imaginary == 5.1;

        ComplexNumber sum = num2.add(num3);

        assert Math.abs(sum.real - 12.5) < DOUBLE_EPSILON;
        assert Math.abs(sum.imaginary - (-5.1)) < DOUBLE_EPSILON;

        ComplexNumber diff = num4.subtract(num3);

        assert Math.abs(diff.real - (-2.5)) < DOUBLE_EPSILON;
        assert Math.abs(diff.imaginary - 5.1012) < DOUBLE_EPSILON;

        ComplexNumber product = num3.multiply(num4);

        assert Math.abs(product.real - 0.00612) < DOUBLE_EPSILON;
        assert Math.abs(product.imaginary - 0.003) < DOUBLE_EPSILON;

        ComplexNumber quotient = num4.divide(num3);

        assert Math.abs(quotient.real - (-1.89709E-4)) < DOUBLE_EPSILON;
        assert Math.abs(quotient.imaginary - 9.29944E-5) < DOUBLE_EPSILON;
    }
}
```

## 4. 커밋, 푸시 그리고 빌드 요청

이건 어떻게 하는지 이제 다 아시죠? :)