다양한 형태의 함수 구현

- 복소수의 덧셈 연산을 함수로 구현하는 방법들

  방법1: c = addComplex(a, b);// 일반 함수로 구현 (실습4)
  방법2: c.add(a, b); // 클래스의 멤버함수로 구현(실습5)
  방법3: c = a.add(b); // 클래스의 멤버함수로 구현
  방법4: c = a.operator+(b); // 연산자 오버로딩
  방법5: c = a + b; // 연산자 오버로딩

- 동일한 연산을 함수로 구현하는 방법에는 여러 가지가 있다.

복소수 클래스

Complex (version 4)
- real : *double* - imag : *double*
+ set(double r, double i) + read(char* msg) + print(char* msg) + add(Complex p, Complex p)

Complex (version 5)
- real : *double* - imag : *double*
+ set(double r, double i) + read(char* msg) + print(char* msg) + add(Complex p, Complex p) + add(Complex p) : *Complex* + operator+(Complex p) : *Complex*

다양한 형태의 함수 구현

- 방법 1:

  c = addComplex(a, b);

```
// 일반 함수로 구현한 복소수 덧셈 연산
// 매개변수 a와 b를 더한 복소수 객체를 만들고 이를 반환
Complex addComplex( Complex a, Complex b )
{
    Complex c;
    c.real = a.real + b.real;
    c.imag = a.imag + b.imag;
    return c;
}
```

- 일반 함수로 구현
  - 함수의 이름:
  - 매개변수:
  - 반환형:

다양한 형태의 함수 구현

- 방법 2:

  c.add(a, b); // Complex::add(…)

```
// Complex 클래스의 멤버 함수로 구현한 복소수 덧셈 연산
// 매개변수 a와 b를 받아 복소수 객체 자신의 값을 채움
// 반환할 자료는 없음
void add( Complex a, Complex b ) {
    real = a.real + b.real;
    imag = a.imag + b.imag;
}
```

- Complex 클래스의 멤버 함수로 구현
  - 함수의 이름:
  - 매개변수:
  - 반환형:

다양한 형태의 함수 구현

- 방법 3:

  c = a.add(b); // Complex::add(…)

```
// Complex 클래스의 멤버 함수로 구현한 복소수 덧셈 연산
// 매개변수 b를 받아 복소수 객체 자신과 b를 더한
// 새로운 복소수 객체를 만들고, 이것을 반환함.
Complex add( Complex b ) {
    Complex c;
    c.real = real + b.real;
    c.imag = imag + b.imag;
    return c;
}
```

- Complex 클래스의 멤버 함수로 구현
  - 함수의 이름:
  - 매개변수:
  - 반환형:

다양한 형태의 함수 구현

- 방법 4:

  c = a.operator+(b); // Complex::operator+ (…)

```
// 연산자 오버로딩으로 구현한 복소수 덧셈 함수.
Complex operator+( Complex b ) {
    Complex c;
    c.real = real + b.real;
    c.imag = imag + b.imag;
    return c;
}
//  Complex operator+( Complex b ) { return add(b); }
```

- Complex 클래스의 멤버 함수로 구현
  - 함수의 이름:
  - 매개변수:
  - 반환형:

다양한 형태의 함수 구현

- 방법 5:

  c = a + b; // Complex::operator+ (…)

```
// 연산자 오버로딩으로 구현한 복소수 덧셈 함수.
Complex operator+( Complex b ) {
    Complex c;
    c.real = real + b.real;
    c.imag = imag + b.imag;
    return c;
}
//  Complex operator+( Complex b ) { return add(b); }
```

- 어떤 방법이 가장 좋아 보이는가?

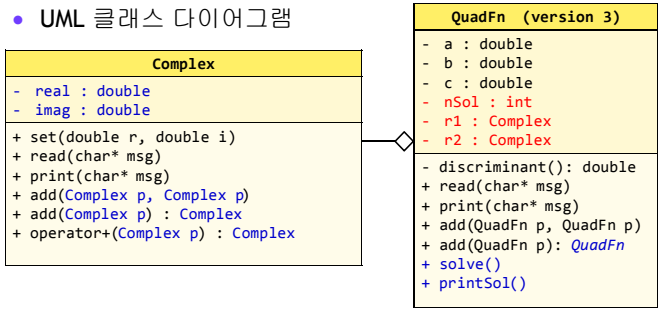
이차방정식 클래스

QuadFn (version 2)
- a : double - b : double - c : double
- discriminant(): double + read(char* msg) + print(char* msg) + add(QuadFn p, QuadFn p) + add(QuadFn p): *QuadFn* + solve(Complex& r1, Complex& r2): *int* + printSol(Complex r1, Complex r2, int n)

QuadFn (version 3)
- a : double - b : double - c : double - nSol : int - r1 : Complex - r2 : Complex
- discriminant(): double + read(char* msg) + print(char* msg) + add(QuadFn p, QuadFn p) + add(QuadFn p): *QuadFn* + solve() + printSol()

이차방정식의 클래스 다이어그램

- UML 클래스 다이어그램



예제: 복소수의 부호 변환 함수

- 복소수의 부호 변환 함수

```
방법1: c = negateComplex(a);
방법2: c.negate();          // 클래스의 멤버함수로 구현
방법3: c = a.negate();      // 클래스의 멤버함수로 구현
방법4: c = a.operator-();   // 연산자 오버로딩
방법5: c = -a;              // 연산자 오버로딩
```


예제: 이차방정식의 함수 구현

- Solve() 함수

- PrintSolution() 함수

Q & A