Assignment

lab 01

Yang Dong Jae

A Data Structure Homework Assignment

September 7, 2023

Problem 1

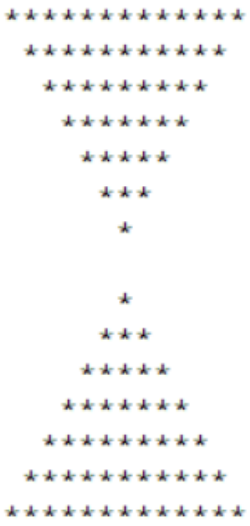
Function class

The Functions class/Type that implements the following functions

(a) Write a function (getFactorial). The factorial of a non-negative integer n is written n! (pronounced "n factorial") and is defined as follows:

$$n! = n \times (n-1) \times (n-2) \times ... \times 1$$

(b) Write a function drawTriangles() that prints the downward and upward triangles



(c) getTriples: A right triangle can have sides that are all integers. The set of three integer values for the sides of a right triangle is called a Pythagorean triple. These three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse. Find all Pythagorean triples for side1, side2 and hypotenuse all no larger than 50.

Solution.

```
class Functions:
    @staticmethod
    def getFactorial(n):
        if n == 0:
            return 1
        else:
            return n * Functions.getFactorial(n - 1)

    @staticmethod
    def drawTriangles(height):
```

```
        if height <= 0:
            print("Height must be a positive integer.")
            return

        for i in range(height, 0, -1):
            spaces = " " * (height - i)
            stars = "*" * (2 * i - 1)
            print(spaces + stars)

        print("")

        for i in range(1, height + 1):
            spaces = " " * (height - i)
            stars = "*" * (2 * i - 1)
            print(spaces + stars)

    @staticmethod
    def getTriples(limit):
        triples = []
        for side1 in range(1, limit + 1):
            for side2 in range(side1, limit + 1):
                hypotenuse = (side1 ** 2 + side2 ** 2) ** 0.5
                if hypotenuse.is_integer() and hypotenuse <= limit:
                    triples.append((side1, side2, int(hypotenuse)))
        return triples

# Result
# 5! = 120
# *********
#  *******
#   *****
#    ***
#     *

#     *
#    ***
#   *****
#  *******
# *********
# (3, 4, 5)
# (5, 12, 13)
# (6, 8, 10)
# (7, 24, 25)
# (8, 15, 17)
# (9, 12, 15)
# (9, 40, 41)
# (10, 24, 26)
# (12, 16, 20)
# (12, 35, 37)
# (14, 48, 50)
# (15, 20, 25)
# (15, 36, 39)
# ...
# (21, 28, 35)
```

```
# (24, 32, 40)
# (27, 36, 45)
# (30, 40, 50)
```

getFactorial function Detail

- 요구사항 : 임의의 정수 n이 주어질 때, n의 Factorial을 구하는 것
- 문제분석: 이 함수는 임의의 정수 n의 팩토리얼을 계산해야 합니다. 여기서 주요한 포인트는 클래스 Functions 내부에서 직접적인 인스턴스 변수에 접근할 필요가 없으므로 staticmethod를 활용하여 정적 메서드로 만들어 최적화할 수 있습니다. 또한, 팩토리얼은 재귀적으로 정의할 수 있으므로 재귀 함수를 사용하여 구현할 수 있습니다.
- 접근방법
  a) 함수를 staticmethod로 정의하여 클래스의 인스턴스를 생성하지 않고도 사용 가능하도록 만듭니다.
  b) Base Case: 만약 n이 0이면 1을 반환합니다. (0! = 1)
  c) Recursive Case: 그렇지 않으면, n * getFactorial(n - 1)을 반환하여 recursivly하게 팩토리얼을 계산합니다.
  d) 이렇게 recursive call을 통해 n부터 1까지의 숫자를 곱하여 최종 팩토리얼 값을 계산합니다.

drawTriangles function detail

- 요구사항 : 주어진 높이(height)에 따라 내려가는(downward) 삼각형과 올라가는(upward) 삼각형을 출력하는 것
- 문제분석: 이 함수는 주어진 높이(height)에 따라 별(*)과 공백(space)을 활용하여 삼각형을 출력해야 합니다. 내려가는 삼각형과 올라가는 삼각형을 각각 출력해야 하며, 높이(height)는 양의 정수여야 합니다
- 접근방법
  a) 주어진 높이(height)가 0 이하인 경우에는 "Height must be a positive integer."라는 에러 메시지를 출력하고 함수를 종료합니다.
  b) 높이(height)에 따라 내려가는 삼각형을 출력합니다.
    (a) 첫 번째 for 루프는 i를 높이부터 1까지 감소시키며 반복합니다.
    (b) 각 줄에서는 i에 따른 공백(spaces)과 별(stars)을 계산하여 출력합니다.
  c) 한 줄을 비웁니다.
  d) 높이(height)에 따라 올라가는 삼각형을 출력합니다.
    (a) 두번째 for 루프는 i를 1부터 높이까지 증가시키며 반복합니다.
    (b) 각 줄에서는 i에 따른 공백(spaces)과 별(stars)을 계산하여 출력합니다.

- 점화식 (Recurrence Relation)
  - 내려가는 삼각형 각 줄의 공백 개수 : height - i
  - 내려가는 삼각형 각 줄의 별 개수 : 2 * i - 1
  - 올라가는 삼각형 각 줄의 공백 개수 : height - i
  - 올라가는 삼각형 각 줄의 별 개수 : 2 * i - 1

getTriples

- 요구사항 : 주어진 한계(limit) 내에서 가능한 모든 피타고라스 삼중 (side1, side2, hypotenuse)을 찾아야 합니다. 여기서 각 변(side)은 주어진 한계(limit) 이하의 값을 가져야 합니다.
- 문제분석 : 이 함수는 피타고라스 정리를 따르는 삼중을 찾는 작업을 수행합니다. 피타고라스 정리에 따르면,

$$a^2 + b^2 = c^2$$

  인 삼각형에서 각 변(side) a, b, c는 양의 정수일 때 유효한 피타고라스 삼중입니다. 주어진 한계(limit) 내에서 가능한 모든 피타고라스 삼중을 찾아야 합니다.
- 접근방법

  a) 주어진 한계(limit) 내에서 가능한 피타고라스 삼중을 저장할 빈 리스트(triples)를 생성합니다.

  b) 두 개의 반복문을 사용하여 side1과 side2를 선택합니다.

  (a) 첫 번째 반복문에서 side1을 1부터 limit까지 반복합니다.

  (b) 두 번째 반복문에서 side2를 side1부터 limit까지 반복합니다. 이렇게 하면 side1 <= side2가 되어 중복을 피할 수 있습니다.

  c) 반복문에서 side1과 side2를 이용하여 hypotenuse(빗변)를 계산합니다. 피타고라스 정리에 따르면 $hypotenuse^2 = side1^2 + side2^2$ 입니다.

  d) 계산된 hypotenuse가 정수이고, 주어진 한계(limit) 이하인 경우, 즉, 'hypotenuse.is_integer()'가 True이고 hypotenuse가 limit 이하인 경우에만 피타고라스 삼중을 리스트(triples)에 추가합니다.

  e) 모든 반복이 완료된 후, 모든 가능한 피타고라스 삼중이 저장된 리스트(triples)를 반환합니다.

Problem 2

Complex class

- A complex number is a number of the form x + yi, where x and y are real numbers and i is the square root of -1. The number x is known as the real part of the complex number, and the number y is known as the imaginary part.
- The operations on complex numbers that are needed for basic computations

- *Addition*: $(x+yi) + (v+wi) = (x+v) + (y+w)i$
- *Multiplication*: $(x + yi) * (v + wi) = (xv - yw) + (yv + xw)i$
- *Magnitude*: $|x + yi| = (x^2 + y^2)^{1/2}$
- *Real part*: $\mathrm{Re}(x + yi) = x$
- *Imaginary part*: $\mathrm{Im}(x + yi) = y$

client operation	*special method*	*description*
`Complex(x, y)`	`__init__(self, re, im)`	*new `Complex` object with value x+yi*
`a.re()`		*real part of a*
`a.im()`		*imaginary part of a*
`a + b`	`__add__(self, other)`	*sum of a and b*
`a * b`	`__mul__(self, other)`	*product of a and b*
`abs(a)`	`__abs__(self)`	*magnitude of a*
`str(a)`	`__str__(self)`	*'x + yi' (string representation of a)*

API for a user-defined `Complex` data type

Solution.

```python
import math

class Complex:
    def __init__(self, x=0, y=0):
        self.re = x  # Real part
        self.im = y  # Imaginary part

    def __str__(self):
        if self.im >= 0:
            return f"{self.re} + {self.im}i"
        else:
            return f"{self.re} - {abs(self.im)}i"

    def __repr__(self):
        return f"Complex({self.re}, {self.im})"

    def __add__(self, other):
        if isinstance(other, Complex):
            return Complex(self.re + other.re, self.im + other.im)
        else:
            raise TypeError("Unsupported operand type for +")
```

```
22
23      def __mul__(self, other):
24          if isinstance(other, Complex):
25              real = self.re * other.re - self.im * other.im
26              imag = self.re * other.im + self.im * other.re
27              return Complex(real, imag)
28          else:
29              raise TypeError("Unsupported operand type for *")
30
31      def __abs__(self):
32          return math.sqrt(self.re ** 2 + self.im ** 2)
33
34      def __eq__(self, other):
35          if isinstance(other, Complex):
36              return self.re == other.re and self.im == other.im
37          else:
38              return False
39
40      def __ne__(self, other):
41          if isinstance(other, Complex):
42              return not self.__eq__(other)
43          else:
44              return True
45
46      def __gt__(self, other):
47          if isinstance(other, Complex):
48              return abs(self) > abs(other)
49          else:
50              raise TypeError("Unsupported operand type for >")
51
52      def __le__(self, other):
53          if isinstance(other, Complex):
54              return not self.__gt__(other)
55          else:
56              raise TypeError("Unsupported operand type for <=")
57
58
59  if __name__ == "__main__":
60      z1 = Complex(3, 4)  # Create a complex number 3 + 4i
61      z2 = Complex(1, -2)  # Create a complex number 1 - 2i
62
63      # Test addition
64      result_add = z1 + z2
65      print("Addition:", result_add)
66
67      # Test multiplication
68      result_mul = z1 * z2
69      print("Multiplication:", result_mul)
70
71      # Test absolute value
72      abs_z1 = abs(z1)
73      print("Absolute Value of z1:", abs_z1)
74
75      # Test equality
```

```
    z3 = Complex(3, 4)  # Create another complex number 3 + 4i
    print("Equality Test (z1 == z3):", z1 == z3)
    print("Equality Test (z1 == z2):", z1 == z2)

    # Test inequality
    print("Inequality Test (z1 != z3):", z1 != z3)
    print("Inequality Test (z1 != z2):", z1 != z2)

    # Test greater than
    z4 = Complex(5, 1)  # Create another complex number 5 + 0i
    print("Greater Than Test (z1 > z4):", z1 > z4)
    print("Greater Than Test (z2 > z1):", z2 > z1)

    # Test less than or equal to
    z5 = Complex(1, 5)  # Create another complex number 0 + 5i
    print("Less Than or Equal To Test (z5 <= z1):", z5 <= z1)
    print("Less Than or Equal To Test (z1 <= z2):", z1 <= z2)

# Result
# Addition: 4 + 2i
# Multiplication: 11 - 2i
# Absolute Value of z1: 5.0
# Equality Test (z1 == z3): True
# Equality Test (z1 == z2): False
# Inequality Test (z1 != z3): False
# Inequality Test (z1 != z2): True
# Greater Than Test (z1 > z4): False
# Greater Than Test (z2 > z1): False
# Less Than or Equal To Test (z5 <= z1): False
# Less Than or Equal To Test (z1 <= z2): False
```

Complex Class detail

- 요구사항 : 복소수(Complex number)는 x + yi 형태의 수로, x와 y는 실수이며 i는 -1의 제곱근을 나타냅니다. 여기서 x는 복소수의 실수 부분(Real part)이고, y는 복소수의 허수 부분(Imaginary part)입니다. 이러한 복소수에 대한 기본 계산을 수행하는 클래스를 구현해야 합니다.
- 문제분석 : 이 클래스는 복소수에 대한 연산을 수행해야 하며, 요구사항에 따라 복소수의 실수 부분과 허수 부분을 나타내는 속성을 갖습니다. 또한, 덧셈, 곱셈, 절댓값, 등호 비교, 부등호 비교 등의 연산을 수행해야 합니다.
- 접근방법
  - '__init__' 메서드: 이 메서드는 객체를 초기화하고 복소수의 실수 부분과 허수 부분을 받아 초기화합니다.
  - '__str__' 메서드: 이 메서드는 객체를 문자열로 표현할 때 사용됩니다. 허수 부분이 양수인 경우는 $x + yi$ 형태로, 음수인 경우는 $x - yi$ 형태로 출력합니다.
  - '__repr__' 메서드: 이 메서드는 객체를 나타내는 공식적인 문자열 표현을 반환합니다.
  - '__add__' 메서드: 이 메서드는 두 복소수 객체를 더합니다.

  - '___mul___' 메서드: 이 메서드는 두 복소수 객체를 곱합니다.
  - '___abs___' 메서드: 이 메서드는 복소수의 절댓값을 반환합니다.
  - '___eq___' 메서드: 이 메서드는 두 복소수 객체가 같은지 여부를 반환합니다.
  - '___ne___' 메서드: 이 메서드는 두 복소수 객체가 다른지 여부를 반환합니다.
  - '___gt___' 메서드: 이 메서드는 복소수 객체를 크기 비교하여 현재 객체가 다른 객체보다 큰지 여부를 반환합니다.
  - '___le___' 메서드: 이 메서드는 복소수 객체를 크기 비교하여 현재 객체가 다른 객체보다 작거나 같은지 여부를 반환합니다.

- Special Method 사용의 효율성 : Special method를 사용하면 객체 지향 프로그래밍의 기본 원칙을 따르며 코드를 읽기 쉽게 만들 수 있습니다. 예를 들어, z1 + z2와 같은 연산을 사용하면 더 직관적이고 가독성이 높아집니다. 또한, special method를 사용하면 연산자 오버로딩을 지원하여 파이썬의 기본 연산자를 사용하여 객체 간의 연산을 수행할 수 있습니다. 이로써 코드 작성 및 이해가 더 쉬워지며, 코드의 일관성과 가독성이 향상됩니다.

Problem 3

Point Class

- Create a Point3D class and test it in the Lab01Test class. Point3D class should contain the following data and function members
- Data members
  x
  y
  z
  3D point coordinates of type double
- Function members
  ___init___()
  ___str___()
  ___repr___()
  setCord(double, double, double) : assigns the values for three coordinates
  length() : returns distance between point P and origin (0,0,0)
  distance(Point3D, Point3D): returns distance between two points
  translate(double, double, double): translates (moves) the point in the given directions

Solution.

```python
import math

class Point3D:
    def __init__(self, x=0.0, y=0.0, z=0.0):
        self.x = x
        self.y = y
        self.z = z

    def __str__(self):
        return f"({self.x}, {self.y}, {self.z})"

    def __repr__(self):
        return f"Point3D({self.x}, {self.y}, {self.z})"

    def setCord(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z

    def length(self):
        return math.sqrt(self.x**2 + self.y**2 + self.z**2)

    @staticmethod
    def distance(p1, p2):
```

```
        dx = p2.x - p1.x
        dy = p2.y - p1.y
        dz = p2.z - p1.z
        return math.sqrt(dx**2 + dy**2 + dz**2)

    def translate(self, a, b, c):
        self.x += a
        self.y += b
        self.z += c

    def __add__(self, other):
        if isinstance(other, Point3D):
            return Point3D(self.x + other.x, self.y + other.y, self.z + other.z)
        else:
            raise TypeError("Unsupported operand type for +")

    def __sub__(self, other):
        if isinstance(other, Point3D):
            return Point3D(self.x - other.x, self.y - other.y, self.z - other.z)
        else:
            raise TypeError("Unsupported operand type for -")

    def __eq__(self, other):
        if isinstance(other, Point3D):
            return self.x == other.x and self.y == other.y and self.z == other.z
        else:
            return False

    def __gt__(self, other):
        if isinstance(other, Point3D):
            return self.length() > other.length()
        else:
            raise TypeError("Unsupported operand type for >")

    def __le__(self, other):
        if isinstance(other, Point3D):
            return not self.__gt__(other)
        else:
            raise TypeError("Unsupported operand type for <=")


if __name__ == "__main__":
    p1 = Point3D(1.0, 2.0, 3.0)
    print("Point p1:", p1)

    p2 = Point3D(4.0, 5.0, 6.0)
    print("Point p2:", p2)

    p1.setCord(7.0, 8.0, 9.0)
    print("Updated Point p1:", p1)

    length_p1 = p1.length()
    print("Length of p1:", length_p1)

```

```python
    distance_p1_p2 = Point3D.distance(p1, p2)
    print("Distance between p1 and p2:", distance_p1_p2)

    p1.translate(1.0, -1.0, 2.0)
    print("Translated Point p1:", p1)

    # Test addition
    result_add = p1 + p2
    print("Addition:", result_add)

    # Test subtraction
    result_sub = p1 - p2
    print("Subtraction:", result_sub)

    # Test equality
    p3 = Point3D(7.0, 8.0, 9.0)  # Create another point with the same coordinates as p1
    print("Equality Test (p1 == p3):", p1 == p3)
    print("Equality Test (p1 == p2):", p1 == p2)

    # Test greater than
    p4 = Point3D(5.0, 0.0, 0.0)  # Create another point
    print("Greater Than Test (p1 > p4):", p1 > p4)
    print("Greater Than Test (p2 > p1):", p2 > p1)

    # Test less than or equal to
    p5 = Point3D(10.0, 10.0, 10.0)  # Create another point
    print("Less Than or Equal To Test (p5 <= p1):", p5 <= p1)
    print("Less Than or Equal To Test (p1 <= p2):", p1 <= p2)


# Result
# Point p1: (1.0, 2.0, 3.0)
# Point p2: (4.0, 5.0, 6.0)
# Updated Point p1: (7.0, 8.0, 9.0)
# Length of p1: 13.92838827718412
# Distance between p1 and p2: 5.196152422706632
# Translated Point p1: (8.0, 7.0, 11.0)
# Addition: (12.0, 12.0, 17.0)
# Subtraction: (4.0, 2.0, 5.0)
# Equality Test (p1 == p3): False
# Equality Test (p1 == p2): False
# Greater Than Test (p1 > p4): True
# Greater Than Test (p2 > p1): False
# Less Than or Equal To Test (p5 <= p1): False
# Less Than or Equal To Test (p1 <= p2): False
```

Point Class detail

- 요구사항
  - ‘Point3D‘ 클래스를 구현하여 3차원 공간에서의 점을 표현한다.
  - 클래스는 다음 기능을 제공해야 한다.
    a) 초기화 메서드(‘___init___‘): x, y, z 좌표를 받아서 점을 초기화한다.

b) 문자열 표현 메서드(‘___str___‘): 좌표를 문자열로 표현한다.

c) 좌표 설정 메서드(‘setCord‘): 좌표를 변경한다.

d) 점의 길이를 계산하는 메서드(‘length‘): 점의 길이(원점에서의 거리)를 계산한다.

e) 두 점 사이의 거리를 계산하는 정적 메서드(‘distance‘): 두 점 사이의 거리를 계산한다.

f) 점을 이동시키는 메서드(‘translate‘): 좌표를 주어진 양만큼 이동시킨다.

g) 덧셈(‘___add___‘), 뺄셈(‘___sub___‘), 동등성(‘___eq___‘), 크기 비교(‘___gt___‘, ‘___le___‘) 연산을 지원해야 한다.

- 문제분석
  - ‘Point3D‘ 클래스는 3차원 공간에서의 점을 표현하며, 좌표(x, y, z)를 다루어야 합니다.
  - 길이를 계산하거나 두 점 사이의 거리를 계산하는데 수학적인 계산이 필요합니다.
  - 연산자 오버로딩을 통해 덧셈, 뺄셈, 동등성 비교, 크기 비교 연산을 지원해야 합니다.
- 접근방법
  - ‘___init___‘ 메서드: x, y, z 좌표를 인스턴스 변수에 저장
  - ‘___str___‘ 메서드: 좌표를 문자열로 포맷하여 반환
  - ‘setCord‘ 메서드: 좌표를 주어진 값으로 설정
  - ‘length‘ 메서드: 좌표의 길이(원점에서의 거리)를 계산하여 반환
  - ‘distance‘ 정적 메서드: 두 점 사이의 거리를 계산하여 반환
  - ‘translate‘ 메서드: 현재 좌표에 주어진 양만큼 이동
  - 연산자 오버로딩 메서드 (‘___add___‘, ‘___sub___‘, ‘___eq___‘, ‘___gt___‘, ‘___le___‘): 해당 연산을 수행하기 위해 필요한 연산을 정의하고, 예외 처리를 통해 잘못된 연산을 방지

Conclusion

- Importance of Class Design: When designing a class, it's crucial to have a clear understanding of the requirements and problem analysis. Based on this, appropriate methods and attributes can be defined. In the cases of the 'Point3D' and 'Complex' classes, the requirements regarding coordinates and complex numbers were clearly understood and effectively abstracted into the classes.
- Effective Use of Special Methods: Proper utilization of special methods, also known as magic methods, can greatly enhance code readability and usability in object-oriented programming. Methods like '___str___', '___repr___', '___add___', and '___eq___' allow for more intuitive implementations of string representation, addition, and equality checks.
- Utilizing Static Methods: Static methods can be beneficial for implementing class-level operations. For instance, the 'distance' method in the 'Point3D' class was implemented as a static method to calculate the distance between two points. This enhances the organization of code and makes it more semantically clear.
- Operator Overloading: Operator overloading allows for more natural and intuitive usage of operators for objects. By overloading operators like addition and subtraction in the 'Point3D' class, it becomes straightforward to add or subtract two points.
- Exception Handling: Ensuring code robustness is crucial. Proper exception handling, such as using 'TypeError' exceptions to prevent invalid operations or type errors, adds a layer of code safety.
- Importance of Test Code: Writing test code is essential when developing classes. Test code helps verify that class methods behave as expected and protects against breaking functionality when making changes to the code.

In summary, in object-oriented programming, it's important to design classes based on requirements and problem analysis. Effective use of special methods, static methods, operator overloading, and proper exception handling can improve code readability and robustness. Additionally, writing test code is crucial to validate class functionality and ensure code reliability.

느낀점

- 클래스 설계의 중요성: 클래스를 설계할 때 명확한 요구사항과 문제 분석이 중요합니다. 이를 토대로 적절한 메서드와 속성을 정의하고 클래스를 구현할 수 있습니다. 'Point3D' 및 'Complex' 클래스의 경우, 좌표 또는 복소수에 대한 요구사항을 명확하게 이해하고 이를 클래스로 잘 추상화하여 구현하였습니다.
- 특수 메서드 활용: 특수 메서드 또는 매직 메서드를 적절하게 활용하면 객체 지향 프로그래밍에서 코드의 가독성과 사용 편의성을 높일 수 있습니다. '___str___', '___repr___', '___add___', '___eq___' 등의 메서드를 사용하여 객체의 문자열 표현, 덧셈, 동등성 비교 등을 보다 직관적으로 구현할 수 있습니다.
- 정적 메서드 활용: 정적 메서드를 사용하여 클래스 수준의 연산을 구현할 수 있습니다. 'Point3D' 클래스에서 'distance' 메서드를 정적 메서드로 구현한 것처럼, 클래스와 관련된 연산을 구현할 때 정적 메서드를 고려할 수 있습니다.
- 연산자 오버로딩의 활용: 연산자 오버로딩을 사용하면 객체 간의 연산을 직관적으로 구현할 수 있습니다. 'Point3D' 클래스에서 덧셈, 뺄셈 연산을 오버로딩하여 두 점을 더하거나 빼는 코드를 간단하게 작성할 수 있었습니다.
- 예외 처리: 코드 안정성을 위해 적절한 예외 처리를 포함하는 것이 중요합니다. 잘못된 연산이나 타입 에러를 방지하기 위해 'TypeError' 예외를 사용하여 처리하는 부분이 코드 안전성을 높이는데 도움이 되었습니다.
- 테스트 코드 작성의 중요성: 클래스를 개발할 때 테스트 코드를 작성하는 것은 매우 중요합니다. 테스트 코드를 통해 클래스의 메서드가 예상대로 작동하는지 확인하고, 코드 변경 시 기능을 파괴하지 않도록 보호할 수 있습니다.

  객체 지향 프로그래밍에서는 요구사항과 문제 분석을 토대로 클래스를 설계하고 특수 메서드, 정적 메서드, 연산자 오버로딩을 활용하여 객체와 연산을 추상화하고 코드의 가독성을 높이는 것이 중요합니다. 또한, 테스트 코드를 작성하여 코드의 안정성을 검증하는 습관도 중요하다는 점을 배웠습니다.