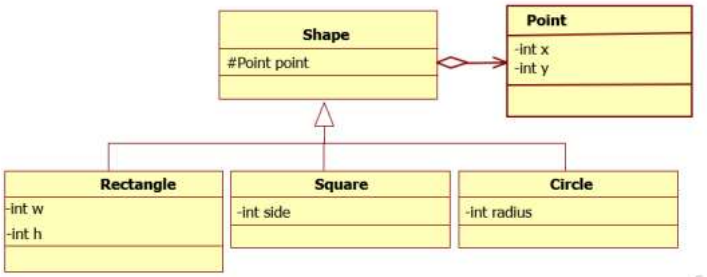
Lab #3: OOP1

22130247 - Trần Ngọc Tân

Task 1. For a given class diagram as follows:



Implements the following methods:

1) Overload common operators (i.e., +, -, *, /, >, >=, <, <=, ==) for Point class
2) Overload common operators (i.e. >, >=, <, <=, ==) for Rectangle, Square, Circle classess
3) *p()*: compute the perimeter of a shape
4) *area()*: compute the area of a shape
5) *distanceToO()*: compute the distance from the point to O
6) *inside(Point p)*: check whether a given point (p) is inside a shape?

```
import math

# Point class
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Point(self.x - other.x, self.y - other.y)

    def __mul__(self, value):
        return Point(self.x * value, self.y * value)

    def __truediv__(self, value):
        return Point(self.x / value, self.y / value)

    def __eq__(self, other):
        return self.x == other.x and self.y == other.y

    def __lt__(self, other):
        return (self.x ** 2 + self.y ** 2) < (other.x ** 2 + other.y ** 2)

    def __le__(self, other):
        return (self.x ** 2 + self.y ** 2) <= (other.x ** 2 + other.y ** 2)

    def __gt__(self, other):
        return (self.x ** 2 + self.y ** 2) > (other.x ** 2 + other.y ** 2)

    def __ge__(self, other):
        return (self.x ** 2 + self.y ** 2) >= (other.x ** 2 + other.y ** 2)

    # Calculate the distance to the origin (0, 0)
    def distanceToO(self):
        return math.sqrt(self.x ** 2 + self.y ** 2)


# Shape class
class Shape:
    def __init__(self, point):
        self.point = point

    def __lt__(self, other):
        return self.area() < other.area()

    def __le__(self, other):
```

```
            return self.area() <= other.area()

        def __gt__(self, other):
            return self.area() > other.area()

        def __ge__(self, other):
            return self.area() >= other.area()

        def __eq__(self, other):
            return self.area() == other.area()

        def p(self):
            pass

        def area(self):
            pass

        def inside(self, p):
            pass


    # Rectangle class
    class Rectangle(Shape):
        def __init__(self, point, width, height):
            super().__init__(point)
            self.width = width
            self.height = height

        def p(self):
            return 2 * (self.width + self.height)

        def area(self):
            return self.width * self.height

        def inside(self, p):
            return (self.point.x <= p.x <= self.point.x + self.width) and (self.point.y <= p.y <= self.point.y + self.height)


    # Square class
    class Square(Rectangle):
        def __init__(self, point, side):
            super().__init__(point, side, side)


    # Circle class
    class Circle(Shape):
        def __init__(self, point, radius):
            super().__init__(point)
            self.radius = radius

        def p(self):
            return 2 * math.pi * self.radius

        def area(self):
            return math.pi * self.radius ** 2

        def inside(self, p):
            return (p.x - self.point.x) ** 2 + (p.y - self.point.y) ** 2 <= self.radius ** 2


    # Ex:
    origin = Point(0,0)
    point1 = Point(3, 4)
    rect = Rectangle(origin, 4, 5)
    square = Square(origin, 3)
    circle = Circle(origin, 2)

    print("Rectangle perimeter:", rect.p())
    print("Rectangle area:", rect.area())
    print("Square perimeter:", square.p())
    print("Square area:", square.area())
    print("Circle perimeter:", circle.p())
    print("Circle area:", circle.area())
    print("Point1 distance to origin:", point1.distanceToO())
    print("Point1 inside Rectangle:", rect.inside(point1))
    print("Point1 inside Circle:", circle.inside(point1))
    print("Point1 inside Square:", square.inside(point1))
```

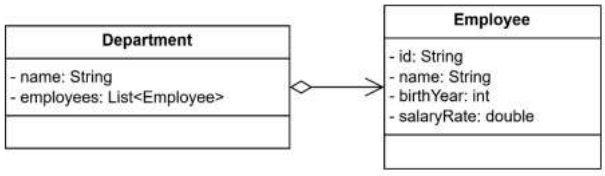
```
Rectangle perimeter: 18
Rectangle area: 20
Square perimeter: 12
Square area: 9
```

```
Circle perimeter: 12.566370614359172
Circle area: 12.566370614359172
Point1 distance to origin: 5.0
Point1 inside Rectangle: True
Point1 inside Circle: False
Point1 inside Square: False
```

```
#Ex:
point1 = Point(3, 4)
point2 = Point(1, 2)

add_result = point1 + point2
print(f"point1 + point2: ({add_result.x}, {add_result.y})")

sub_result = point1 - point2
print(f"point1 - point2: ({sub_result.x}, {sub_result.y})")

mul_result = point1 * 2
print(f"point1 * 2: ({mul_result.x}, {mul_result.y})")

div_result = point1 / 2
print(f"point1 / 2: ({div_result.x}, {div_result.y})")

eq_result = point1 == point2
print(f"point1 == point2: {eq_result}")

gt_result = point1 > point2
print(f"point1 > point2: {gt_result}")

ge_result = point1 >= point2
print(f"point1 >= point2: {ge_result}")

lt_result = point1 < point2
print(f"point1 < point2: {lt_result}")

le_result = point1 <= point2
print(f"point1 <= point2: {le_result}")
```

```
point1 + point2: (4, 6)
point1 - point2: (2, 2)
point1 * 2: (6, 8)
point1 / 2: (1.5, 2.0)
point1 == point2: False
point1 > point2: True
point1 >= point2: True
point1 < point2: False
point1 <= point2: False
```

Task 2. For a given class diagram (in Java):



Implements the following methods in **Department** class:

1) *countEmployees(int year)*: determine the number of employees having birth year equals to a given year
2) *findOldestEmployee()*: find the oldest employee in the department
3) *statByBirthYear()*: to make a statistic on the number of employees by birth year, the key is birth year and the value is the number of students.

```
class Employee:
    def __init__(self, id, name, birthYear, salaryRate):
        self.id = id
        self.name = name
        self.birthYear = birthYear
        self.salaryRate = salaryRate

class Department:
    def __init__(self,name,employees=[]):
        self.name = name
        self.employees = employees

    def add_employee(self, employee):
```

```
            self.employees.append(employee)

        def count_employees(self, year):
            count = 0
            for employee in self.employees:
                if employee.birthYear == year:
                    count += 1
            return count

        def findOldestEmployee(self):
            oldest_employee = None
            for employee in self.employees:
                if oldest_employee is None or employee.birthYear < oldest_employee.birthYear:
                    oldest_employee = employee
            return oldest_employee

        def statByBirthYear(self):
            birth_year_stats = {}
            for employee in self.employees:
                if employee.birthYear in birth_year_stats:
                    birth_year_stats[employee.birthYear] += 1
                else:
                    birth_year_stats[employee.birthYear] = 1
            return birth_year_stats
```

```
#Ex:


employee1 = Employee("E001", "Alice", 1985, 50000)
employee2 = Employee("E002", "Bob", 1990, 60000)
employee3 = Employee("E003", "Charlie", 1985, 55000)
employee4 = Employee("E004", "Diana", 1995, 70000)

dept = Department("Samsung")
dept.add_employee(employee1)
dept.add_employee(employee2)
dept.add_employee(employee3)
dept.add_employee(employee4)


print("1: Count of employees born in 1985:", dept.count_employees(1985))
oldest_employee = dept.findOldestEmployee()
print("2: Oldest employee:", oldest_employee.name if oldest_employee else "No employees")
print("3: Statistics by birth year:", dept.statByBirthYear())
```

```
1: Count of employees born in 1985: 4
2: Oldest employee: Alice
3: Statistics by birth year: {1985: 4, 1990: 2, 1995: 2}
```

Task 3. The bookstore has two types of publications (references and magazines). Each publication has the following information: **title**, **number of pages**, **year of publication**, **author**, and **price**. Magazines have additional attributes: **name**. Reference books have additional attributes: **field** (e.g., medicine, sports, education, etc.) and **chapters**. Each chapter contains information such as **title** and **number of pages**. The bookstore has a list of publications.

Implement the following methods:

1) Determine the type of each publication (Magazine or Reference).
2) Whether a publication is a magazine and was published 10 years ago (from 2024).
3) Whether two publications are of the same type and by the same author.
4) Compute the cost of all publications in the bookstore.
5) Find the reference book with the chapter containing the most pages.
6) Check whether a magazine is in the bookstore based on its name.
7) Find all magazines published in a given year.

```
from typing import List

class Publication:
    def __init__(self, title, num_pages, year, author, price):
```

```
            self.title = title
            self.num_pages = num_pages
            self.year = year
            self.author = author
            self.price = price

    class Magazine(Publication):
        def __init__(self, title, num_pages, year, author, price, name):
            super().__init__(title, num_pages, year, author, price)
            self.name = name

        def __str__(self):
            return f"Magazine(Name: {self.name}, Title: {self.title}, Year: {self.year}, Author: {self.author}, Pages: {self.num_pages}, Pri

    class Chapter:
        def __init__(self, title, num_pages):
            self.title = title
            self.num_pages = num_pages

    class ReferenceBook(Publication):
        def __init__(self, title, num_pages, year, author, price, field, chapters: List[Chapter]):
            super().__init__(title, num_pages, year, author, price)
            self.field = field
            self.chapters = chapters
        def __str__(self):
            return f"ReferenceBook(Title: {self.title}, Field: {self.field}, Year: {self.year}, Author: {self.author}, Pages: {self.num_page
    class Bookstore:
        def __init__(self, publications: List[Publication]):
            self.publications = publications

        # 1: Determine the type of each publication
        def get_publication_type(self, publication):
            if isinstance(publication, Magazine):
                return "Magazine"
            elif isinstance(publication, ReferenceBook):
                return "Reference"
            return "Unknown"

        # 2: Check if a publication is a magazine published 10 years ago
        def is_old_magazine(self, publication, current_year=2024):
            return isinstance(publication, Magazine) and (current_year - publication.year >= 10)

        # 3: Check if two publications are of the same type and by the same author
        def is_same_type_and_author(self, pub1, pub2):
            return (type(pub1) == type(pub2)) and (pub1.author == pub2.author)

        # 4: Calculate the total cost of all publications in the bookstore
        def total_cost(self):
            return sum(pub.price for pub in self.publications)

        # 5: Find the reference book with the chapter containing the most pages
        def reference_with_max_chapter_pages(self):
            max_pages = 0
            max_book = None
            for pub in self.publications:
                if isinstance(pub, ReferenceBook):
                    for chapter in pub.chapters:
                        if chapter.num_pages > max_pages:
                            max_pages = chapter.num_pages
                            max_book = pub
            return max_book

        # 6: Check if a magazine exists in the bookstore by name
        def has_magazine(self, name):
            for pub in self.publications:
                if isinstance(pub, Magazine) and pub.name == name:
                    return True
            return False

        # 7: Find all magazines published in a given year
        def magazines_published_in_year(self, year):
            return [pub for pub in self.publications if isinstance(pub, Magazine) and pub.year == year]

    # EX:
    chapter1 = Chapter("Chapter 1", 15)
    chapter2 = Chapter("Chapter 2", 20)
    magazine1 = Magazine("Magazine One", 50, 2013, "Author A", 10, "Tech Today")
    magazine2 = Magazine("Magazine Two", 60, 2015, "Author C", 12, "Health Matters")
    ref_book = ReferenceBook("Reference Book One", 300, 2018, "Author B", 30, "Education", [chapter1, chapter2])
    bookstore = Bookstore([magazine1, magazine2, ref_book])

    # Test 1: Determine the publication type
```

```
print("Method 1:", bookstore.get_publication_type(magazine1))

# Test 2: Check if a publication is a magazine and was published 10 years ago
print("Method 2:", bookstore.is_old_magazine(magazine1))

# Test  3: Check if two publications are of the same type and by the same author
print("Method 3:", bookstore.is_same_type_and_author(magazine1, magazine2))

# Test 4: Calculate the total cost of all publications
print("Method 4:", bookstore.total_cost())

# Test 5: Find the reference book with the chapter that has the most pages
print("Method 5:", bookstore.reference_with_max_chapter_pages())

# Test 6: Check if a magazine exists by name
print("Method 6:", bookstore.has_magazine("Tech Today"))

# Test 7: Find all magazines published in the year 2013
for magazine in bookstore.magazines_published_in_year(2013):
    print("Method 7:",magazine)
```

```
Method 1: Magazine
Method 2: True
Method 3: False
Method 4: 52
Method 5: ReferenceBook(Title: Reference Book One, Field: Education, Year: 2018, Author: Author B, Pages: 300, Price: 30)
Method 6: True
Method 7: Magazine(Name: Tech Today, Title: Magazine One, Year: 2013, Author: Author A, Pages: 50, Price: 10)
```