Homework No. 04

Due: 23:59, 03 June, 2023

Max points: 100

Rules

- **No late homeworks.** A penalty of 10 points is applied for each day.
- **No plagiarism.** Collaboration is encouraged, but copying someone else's work without proper attribution is not admitted and invalidates the submission. A penalty is applied to all parties included.

Submission procedure

- Each problem solution should be saved in a separate file. The following naming convention should be used: `problem{number}.{extension}`. For example, `problem1.py` or `problem1.pdf`.
- At the start of each file, homework number, student full name and problem number should be mentioned. For example:

```
"""
Homework 4
Name: John Doe
Problem 1
"""
```

- Solution files should be uploaded to YSU Moodle. Alternatively, you can commit your solutions to a Git repository and provide the repository URL on Moodle.

Problem 1 [25 points]

Write a `BankAccount` class in Python, which models a bank account of a customer. The class should consist of the following components:

- **Properties**
  - `id`: A unique identifier for an account.
  - `name`: The full name of a customer.
  - `balance`: The balance of an account, which should be 0 by default.

  Only getters should be defined for the properties.
- **Methods**
  - It should be possible to initialize an instance either with initial balance or without initial balance.
  - Friendly string representation for an account should be implemented.
  - `deposit(amount)`: adds the given amount to the current balance.
  - `withdraw(amount)`: subtracts the given amount from the current balance. If there are insufficient funds, it should raise an error (`ValueError` can be used).
  - `transfer_to(another_account, amount)`: transfers the given amount from the current account to the given account. If there are insufficient funds, it should raise an error (`ValueError` can be used).

Example

```
class BankAccount:
    pass


account_1 = BankAccount(1, "John Doe")
account_2 = BankAccount(2, "Jane Dane", 1000)

print(account_1) # BankAccount(id=1, name="John Doe", balance=0)
print(account_2) # BankAccount(id=2, name="Jane Dane", balance=1000)

account_1.deposit(500)
print(account_1) # BankAccount(id=1, name="John Doe", balance=500)
account_1.withdraw(600) # raises an error

account_2.transfer_to(account_1, 250)
print(account_1) # BankAccount(id=1, name="John Doe", balance=750)
print(account_2) # BankAccount(id=2, name="Jane Dane", balance=750)
account_2.transfer_to(account_1, 800) # raises an error
```

Problem 2 [25 points]

Write a superclass `Shape` and its subclasses `Circle` and `Rectangle` in Python.

- The `Shape` class should consist of the following components:
  - **Properties**
    - `color`: A color that indicates the color of a shape.
    - `is_filled`: A boolean flag that indicates if a shape is filled or not.

    Getters and setters should be defined for the properties.
  - **Methods**
    - It should be possible to initialize an instance by providing a color and whether the shape is filled or not.
    - Friendly string representation for a shape should be implemented.
    - `calculate_area()`: It should raise an error (`NotImplementedError` can be used).
    - `calculate_perimeter()`: It should raise an error (`NotImplementedError` can be used).
- The `Circle` class derives from the `Shape` class and should consist of the following components:
  - **Properties**
    - `radius`: The circle's radius.

    Getters and setters should be defined for the properties.
  - **Methods**
    - It should be possible to initialize an instance by providing a radius, a color, and whether the circle is filled or not.
    - Friendly string representation for a circle should be implemented.
    - `calculate_area()`: It should return the area of a circle.
    - `calculate_perimeter()`: It should return the perimeter of a circle.
- The `Rectangle` class derives from the `Shape` class and should consist of the following components:
  - **Properties**
    - `width`: The rectangle's width.
    - `length`: The rectangle's length.

    Getters and setters should be defined for the properties.
  - **Methods**
    - It should be possible to initialize an instance by providing a width, length, a color, and whether the circle is filled or not.
    - Friendly string representation for a rectangle should be implemented.
    - `calculate_area()`: It should return the area of a rectangle.
    - `calculate_perimeter()`: It should return the perimeter of a rectangle.

Example

```python
class Shape:
    pass

class Circle(Shape):
    pass

class Rectangle(Shape):
    pass

shape = Shape("red", True)
print(shape) # Shape(color="red", is_filled=True)

shape.calculate_area() # raises an error
shape.calculate_perimeter() # raises an error

circle = Circle("black", False, 3)
print(circle) # Circle(color="black", is_filled=False, radius=3)
print(circle.calculate_area()) # 28.27
print(circle.calculate_perimeter()) # 18.85

rectangle = Rectangle("green", True, 3, 4)
print(rectangle) # Rectangle(color="green", is_filled=True, width=3, length=4)
print(rectangle.calculate_area()) # 12
print(rectangle.calculate_perimeter()) # 14
```

Problem 3 [25 points]

Write `Point` and `Triangle` classes in Python.

- The `Point` class should consist of the following components:
  - **Properties**
    - `x`: The x-coordinate of a point.
    - `y`: The y-coordinate of a point.

    Getters and setters should be defined for the properties.
  - **Methods**
    - It should be possible to initialize an instance by providing `x` and `y` coordinates.
    - Friendly string representation for a point should be implemented.
    - `get_xy()`: It should return a tuple of `x` and `y` coordinates.
    - `set_xy(x, y)`: It should change the `x` and `y` coordinates.
    - `distance_from_coordinates(x, y)`: It should return the Euclidean distance between the current point and the point at `(x, y)`.
    - `distance_from_point(another_point)`: It should return the Euclidean distance between the current point and the other point.
    - `abs()`: Point's absolute value should return the Euclidean distance of the point from the origin.
- The `Triangle` class should consist of the following components:
  - **Properties**
    - `vertex1`: The first vertex of a triangle modelled by `Point`.
    - `vertex2`: The second vertex of a triangle modelled by `Point`.
    - `vertex3`: The third vertex of a triangle modelled by `Point`.

    Getters and setters are not needed for the properties.
  - **Methods**
    - It should be possible to initialize an instance by providing `x` and `y` coordinates for all three vertices. Optionally, a feature to initialize an instance by three `Point` objects can be added.
    - Friendly string representation for a triangle should be implemented.
    - `calculate_perimeter()`: It should return the perimeter of a triangle.
    - `get_type()`: It should return the type of a triangle (equilateral, isosceles or scalene).

Example

```
class Point:
    pass

class Triangle:
    pass


point1 = Point(1, 2)
point2 = Point(1, 2)

print(point2) # Point(x=1, y=2)
print(point2.get_xy()) # (1, 2)

point2.set_xy(3, 4)
print(point2) # Point(x=3, y=4)

print(point1.distance_from_coordinates(3, 4)) # 2.83
print(point1.distance_from_point(point2)) # 2.83

print(abs(point1)) # 2.24
print(abs(point2)) # 5

triangle = Triangle(0, 0, 1, 1, 2, 2) # raises an error (No such triangle exists)

triangle = Triangle(0, 0, 0, 4, 2, 0)
print(triangle) # Triangle(vertex1=Point(0, 0), vertex2=Point(0, 4), vertex1=Point(2, 0))

print(triangle.calculate_perimeter()) # 10.47
print(triangle.get_type()) # scalene
```

Problem 4 [25 points]

Write a `Complex` class in Python. The class should consist of the following components:

- **Properties**
  - `real`: The real part of a complex number.
  - `imaginary`: The imaginary part of a complex number.

  Getters and setters are not required.
- **Methods**
  - It should be possible to initialize an instance by providing real and imaginary parts of a complex number.
  - Friendly string representation for a complex number should be implemented.
  - `+`: Addition of two complex numbers should be implemented.
  - `-`: Subtraction of two complex numbers should be implemented.
  - `*`: Multiplication two complex numbers should be implemented.
  - `/`: Division of two complex numbers should be implemented.
  - `==`: Equality checks if two complex numbers are equal.
  - `abs()`: Absolute value of a complex number should return the magnitude of a complex number.

Example

```python
class Complex:
    pass


c1 = Complex(1, 2)
c2 = Complex(3, 4)
c3 = Complex(1, 2)

print(c1) # Complex(real=1, imaginary=2)
print(c1 + c2) # Complex(real=4, imaginary=6)
print(c1 - c2) # Complex(real=-2, imaginary=-2)
print(c1 * c2) # Complex(real=-5, imaginary=10)
print(c1 / c2) # Complex(real=0.44, imaginary=0.08)
print(c1 == c2) # False
print(c1 == c3) # True
```