Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Департамент перспективной инженерии

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
дисциплины «Объектно-ориентированное программирование»
Вариант №1

Выполнила:
Беседина Инга Олеговна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника», очная
форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., канд. технических
наук, доцент, доцент департамента
цифровых, робототехнических систем
и электроники института
перспективной инженерии

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Наследование и полиморфизм в языке Python

Цель: Приобретение навыков по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x

Ход работы

Пример 1. Наследование класса

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

class Figure:

    def __init__(self, color):
        self.__color = color

    @property
    def color(self):
        return self.__color

    @color.setter
    def color(self, c):
        self.__color = c


class Rectangle(Figure):

    def __init__(self, width, height, color):
        super().__init__(color)
        self.__width = width
        self.__height = height

    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, w):
        if w > 0:
            self.__width = w
        else:
            raise ValueError

    @property
    def height(self):
        return self.__height

    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
```

```
            raise ValueError

    def area(self):
        return self.__width * self.__height


if __name__ == '__main__':
    rect = Rectangle(10, 20, 'green')

    print(rect.width)
    print(rect.height)
    print(rect.color)

    rect.color = 'red'
    print(rect.color)
```

```
10
20
green
red
```

Рисунок 1. Результат выполнения программы

Пример 2. Полиморфизм

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

class Figure:

    def __init__(self, color):
        self.__color = color

    @property
    def color(self):
        return self.__color

    @color.setter
    def color(self, c):
        self.__color = c

    def info(self):
        print('Figure')
        print('Color: ' + self.__color)


class Rectangle(Figure):

    def __init__(self, width, height, color):
        super().__init__(color)
        self.__width = width
        self.__height = height
```

```
    @property
    def width(self):
        return self.__width

    @width.setter
    def width(self, w):
        if w > 0:
            self.__width = w
        else:
            raise ValueError

    @property
    def height(self):
        return self.__height

    @height.setter
    def height(self, h):
        if h > 0:
            self.__height = h
        else:
            raise ValueError

    def area(self):
        return self.__width * self.__height

    def info(self):
        print('Rectangle')
        print('Color: ', self.color)
        print("Width: " + str(self.width))
        print("Height: " + str(self.height))
        print("Area: " + str(self.area()))


if __name__ == '__main__':
    fig = Figure('orange')
    fig.info()

    rect = Rectangle(10, 20, 'green')
    rect.info()
```

```
Figure
Color: orange
Rectangle
Color:  green
Width: 10
Height: 20
Area: 200
```

Рисунок 2. Результат выполнения программы

Пример 3. Абстрактные классы в Python

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from abc import ABC, abstractmethod


class Polygon(ABC):

    @abstractmethod
    def noofsides(self):
        pass


class Triangle(Polygon):

    def noofsides(self):
        print('I have 3 sides')


class Pentagon(Polygon):

    def noofsides(self):
        print("I have 5 sides")


class Hexagon(Polygon):

    def noofsides(self):
        print("I have 6 sides")


class Quadrilateral(Polygon):

    def noofsides(self):
        print("I have 4 sides")

if __name__ == '__main__':
    R = Triangle()
    R.noofsides()

    K = Quadrilateral()
    K.noofsides()

    R = Pentagon()
    R.noofsides()

    K = Hexagon()
    K.noofsides()
```

```
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
```

Рисунок 3. Результат выполнения программы

9. Разработайте программу по следующему описанию.

В некой игре-стратегии есть солдаты и герои. У всех есть свойство, содержащее уникальный номер объекта, и свойство, в котором хранится принадлежность команде. У солдат есть метод "иду за героем", который в качестве аргумента принимает объект типа "герой". У героев есть метод увеличения собственного уровня.

В основной ветке программы создается по одному герою для каждой команды. В цикле генерируются объекты-солдаты. Их принадлежность команде определяется случайно. Солдаты разных команд добавляются в разные списки.

Измеряется длина списков солдат противоборствующих команд и выводится на экран. У героя, принадлежащего команде с более длинным списком, увеличивается уровень.

Отправьте одного из солдат первого героя следовать за ним. Выведите на экран идентификационные номера этих двух юнитов.

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import random


class Unit:
    def __init__(self, unit_id, team):
        self.__unit_id = unit_id
        self.__team = team

    @property
    def unit_id(self):
        return self.__unit_id

    @unit_id.setter
    def unit_id(self, unit_id):
        self.__unit_id = unit_id

    @property
    def team(self):
        return self.__team

    @team.setter
    def team(self, team):
        self.__team = team


class Soldier(Unit):
    def __init__(self, unit_id, team):
```

```
        super().__init__(unit_id, team)

    def follow_hero(self, hero):
        if isinstance(hero, Hero):
            return f'Солдат {self.unit_id} следует за героем {hero.unit_id}'
        else:
            return ValueError


class Hero(Unit):
    def __init__(self, unit_id, team, level=1):
        super().__init__(unit_id, team)
        self.__level = level

    @property
    def level(self):
        return self.__level

    def level_up(self):
        self.__level += 1
        return f'Уровень героя {self.unit_id}: {self.level}'


def main():
    hero1 = Hero(1, 'A')
    hero2 = Hero(2, 'B')

    soldiers_team_a = []
    soldiers_team_b = []

    for i in range(10):
        team = random.choice(['A', 'B'])
        soldier = Soldier(i + 3, team)
        if team == 'A':
            soldiers_team_a.append(soldier)
        if team == 'B':
            soldiers_team_b.append(soldier)

    len_a = len(soldiers_team_a)
    len_b = len(soldiers_team_b)

    print(f'Команда A: {len_a} солдат')
    print(f'Команда B: {len_b} солдат')

    if len_a > len_b:
        print(hero1.level_up())
    else:
        print(hero2.level_up())

    print('Номера солдат команды A', end=': ')
    for soldier in soldiers_team_a:
        print(soldier.unit_id, end=', ')
```

```
    print()
    print(soldiers_team_a[random.randint(0, len_a-1)].follow_hero(hero1))


if __name__ == "__main__":
    main()
```

```
Команда A: 3 солдат
Команда B: 7 солдат
Уровень героя 2: 2
Номера солдат команды A: 3, 9, 11,
Солдат 9 следует за героем 1
```

Рисунок 4. Результат выполнения программы

Индивидуальные задания:

1. Создать базовый класс Car (машина), характеризуемый торговой маркой (строка), числом цилиндров, мощностью. Определить методы переназначения и изменения мощности. Создать производный класс Lorry (грузовик), характеризуемый также грузоподъемностью кузова. Определить функции переназначения марки и изменения грузоподъемности.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


class Car:
    def __init__(self, brand, cylinders, power):
        self.__brand = brand
        self.__cylinders = cylinders
        self.__power = power

    def __str__(self):
        return f"Car(brand: {self.__brand}, cylinders: {self.__cylinders}, power:
{self.__power}"

    @property
    def brand(self):
        return self.__brand

    @brand.setter
    def brand(self, new_brand):
        self.__brand = new_brand

    @property
    def cylinders(self):
        return self.__cylinders

    @property
    def power(self):
        return self.__power
```

```
    @power.setter
    def power(self, new_power):
        self.__power = new_power


class Lorry(Car):
    def __init__(self, brand, cylinders, power, load_capacity):
        super().__init__(brand, cylinders, power)
        self.__load_capacity = load_capacity

    def __str__(self):
        return f"Lorry(brand: {self.brand}, cylinders: {self.cylinders}, power:
{self.power}, load_capacity: {self.load_capacity})"

    @property
    def load_capacity(self):
        return self.__load_capacity

    @load_capacity.setter
    def load_capacity(self, new_load_capacity):
        self.__load_capacity = new_load_capacity


def main():
    car = Car('Toyota', 4, 150)
    print(car)

    car.power = 180
    print(car)

    lorry = Lorry('Volvo', 6, 400, 10)
    print(lorry)

    lorry.load_capacity = 12
    lorry.brand = 'Scania'
    print(lorry)


if __name__ == "__main__":
    main()
```

```
Car(brand: Toyota, cylinders: 4, power: 150
Car(brand: Toyota, cylinders: 4, power: 180
Lorry(brand: Volvo, cylinders: 6, power: 400, load_capacity: 10)
Lorry(brand: Scania, cylinders: 6, power: 400, load_capacity: 12)
```

Рисунок 5. Результат выполнения программы

1. Создать абстрактный базовый класс Figure с абстрактными методами вычисления площади и периметра. Создать производные классы: Rectangle (прямоугольник), Circle (круг), Trapezium (трапеция) со своими функциями площади и периметра. Самостоятельно определить, какие поля необходимы, какие из них можно задать в базовом классе, а какие — в производных. Площадь трапеции:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from abc import ABC, abstractmethod
from math import pi


class Figure(ABC):
    def __init__(self, h) -> None:
        self.h = h

    @abstractmethod
    def perimeter(self):
        pass

    @abstractmethod
    def square(self):
        pass


class Rectangle(Figure):
    def __init__(self, h, a) -> None:
        super().__init__(h)
        self.a = a

    def perimeter(self):
        return 2*(self.h+self.a)

    def square(self):
        return self.h * self.a


class Circle(Figure):
    def __init__(self, h) -> None:
        super().__init__(h)

    def perimeter(self):
        return pi*self.h

    def square(self):
        return pi*(self.h/2)**2


class Trapezium(Figure):
    def __init__(self, h, a, b, c, d) -> None:
        super().__init__(h)
```

```
        self.a = a
        self.b = b
        self.c = c
        self.d = d

    def perimeter(self):
        return self.a + self.b + self.c + self.d

    def square(self):
        return ((self.a + self.b)*self.h)/2


def main():
    rect = Rectangle(30, 50)
    print(f"Периметр прямоугольника: {rect.perimeter()}")
    print(f"Площадь прямоугольника: {rect.square()}")

    circ = Circle(30)
    print(f"Периметр круга: {circ.perimeter()}")
    print(f"Площадь круга: {circ.square()}")

    trap = Trapezium(8, 14, 26, 10, 10)
    print(f"Периметр трапеции: {trap.perimeter()}")
    print(f"Площадь трапеции: {trap.square()}")


if __name__ == "__main__":
    main()
```

```
Периметр прямоугольника: 160
Площадь прямоугольника: 1500
Периметр круга: 94.24777960769379
Площадь круга: 706.8583470577034
Периметр трапеции: 60
Площадь трапеции: 160.0
```

Рисунок 6. Результат выполнения программы

Контрольные вопросы:

1. Наследование — это механизм, позволяющий создавать новый класс на основе существующего, унаследовав его свойства и методы. В Python это реализуется с помощью указания родительского класса в скобках при объявлении нового класса

2. Полиморфизм — это возможность использования одного интерфейса для работы с различными типами объектов. В Python это

достигается за счет динамической типизации, где методы могут иметь одинаковые имена, но разные реализации в разных классах

3. "Утиная" типизация — это концепция, согласно которой тип объекта определяется не его классом, а тем, какие методы и свойства он имеет. Если объект "ведет себя" как нужный тип (например, имеет метод quack, значит, это "утка"), то он может использоваться как этот тип

4. Модуль abc (Abstract Base Classes) используется для определения абстрактных базовых классов. Он позволяет создавать интерфейсы с абстрактными методами, которые должны быть реализованы в производных классах

5. Чтобы сделать метод абстрактным, нужно использовать декоратор @abstractmethod из модуля abc

6. Для создания абстрактного свойства можно использовать декоратор @abstractproperty

7. Функция isinstance используется для проверки, является ли объект экземпляром определенного класса или его производного класса. Это полезно для проверки типов во время выполнения программы

Вывод: В ходе выполнения лабораторной работы были приобретены навыки по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x