

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

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

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

Выполнил:
Епифанов Алексей Александрович
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных систем
», очная форма обучения

(подпись)

Проверил:
Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

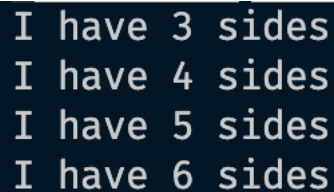
Ставрополь, 2024 г.

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.

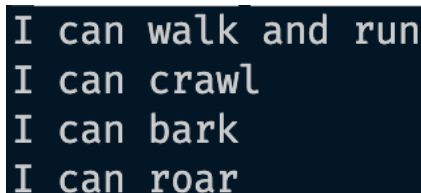
Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.
2. Проработал примеры:



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

Рисунок 1. Результат работы примера 2



```
I can walk and run  
I can crawl  
I can bark  
I can roar
```

Рисунок 2. Результат

3. Выполнил общее задание: 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.

```
Герой 1 добавлен в команду Red
Герой 2 добавлен в команду Blue
Солдат 3 добавлен в команду Red
Солдат 4 добавлен в команду Red
Солдат 5 добавлен в команду Red
Солдат 6 добавлен в команду Red
Солдат 7 добавлен в команду Blue
Солдат 8 добавлен в команду Red
Солдат 9 добавлен в команду Red
Солдат 10 добавлен в команду Red
Солдат 11 добавлен в команду Red
Солдат 12 добавлен в команду Red
Солдаты команды Red: 9
Солдаты команды Blue: 1
Герой 1 из команды Red повысил уровень до 2
Солдат 3 идет за героем 1 из команды Red
```

Рисунок 3. Результат работы программы общего задания

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

import random

class Unit:
    __id_counter = 1

    def __init__(self, team: str):
        self.id = Unit.__id_counter
        Unit.__id_counter += 1
        self.team = team

class Soldier(Unit):
    def __init__(self, team: str):
        super().__init__(team)

    def follow_hero(self, hero):
        """Солдат следует за героем"""
        if isinstance(hero, Hero):
            print(
                f"Солдат {self.id} идет за героем "
                f"{hero.id} из команды {hero.team}"
            )

# Класс героя
```

```

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

    def level_up(self):
        """Увеличивает уровень героя"""
        self.level += 1
        print(
            f"Герой {self.id} из команды "
            f"{self.team} повысил уровень до {self.level}"
        )

# Класс команды
class Team:
    def __init__(self, name: str):
        self.name = name
        self.soldiers = [] # Список солдат команды
        self.hero = None # Герой команды

    def add_hero(self):
        """Добавляет героя в команду"""
        self.hero = Hero(self.name)
        print(f"Герой {self.hero.id} добавлен в команду {self.name}")

    def add_soldier(self):
        """Добавляет солдата в команду"""
        soldier = Soldier(self.name)
        self.soldiers.append(soldier)
        print(f"Солдат {soldier.id} добавлен в команду {self.name}")

    def get_soldier_count(self):
        """Возвращает количество солдат в команде"""
        return len(self.soldiers)

    def level_up_hero(self):
        """Увеличивает уровень героя команды"""
        if self.hero:
            self.hero.level_up()

    def follow_hero(self, index: int):
        """Отправляет первого солдата следовать за героем"""
        if self.soldiers and self.hero:
            first_soldier = self.soldiers[index]
            first_soldier.follow_hero(self.hero)
            return first_soldier.id, self.hero.id
        return None, None

def main():
    # Создаем команды
    team_red = Team("Red")
    team_blue = Team("Blue")

    team_red.add_hero()
    team_blue.add_hero()

    for _ in range(10):
        team = random.choice([team_red, team_blue])
        team.add_soldier()

    print(f"Солдаты команды {team_red.name}: {team_red.get_soldier_count()}")

```

```

print(f"Солдаты команды {team_blue.name}: {team_blue.get_soldier_count()}")

if team_red.get_soldier_count() > team_blue.get_soldier_count():
    team_red.level_up_hero()
elif team_blue.get_soldier_count() > team_red.get_soldier_count():
    team_blue.level_up_hero()
else:
    print("Количество солдат в обеих командах одинаково.")

team_red.follow_hero(0)

if __name__ == "__main__":
    main()

```

4. Выполнил индивидуальное задание 1 вариант 9: Создать класс Pair (пара чисел); определить методы изменения полей и вычисления произведения чисел. Определить производный класс RightAngled с полями-катетами. Определить методы вычисления гипотенузы и площади треугольника.

Произведение чисел (2, 3) равно 6
 Гипотенуза треугольника с катетами (3, 4) равна 5.0
 Площадь треугольника с катетами (3, 4) равна 6.0

Рисунок 4. Результат работы программы индивидуального задания 1

```

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

from math import sqrt

class Pair:
    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def multiply(self):
        return self.x * self.y

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

class RightAngled(Pair):
    def __init__(self, x: float, y: float):
        super().__init__(x, y)

    @property
    def hypotenuse(self):
        return sqrt(self.x**2 + self.y**2)

    @property
    def area(self):
        return self.multiply() / 2

```

```

if __name__ == "__main__":
    pair = Pair(2, 3)
    print(f"Произведение чисел {pair} равно {pair.multiply()}")

    first_triangle = RightAngled(3, 4)
    print(
        "Гипотенуза треугольника с катетами "
        f"{first_triangle} равна {first_triangle.hypotenuse}"
    )
    print(
        "Площадь треугольника с катетами "
        f"{first_triangle} равна {first_triangle.area}"
    )

```

5. Выполнил индивидуальное задание 2 вариант 9: Создать абстрактный базовый класс Pair с виртуальными арифметическими операциями. Реализовать производные классы Complex (комплексное число) и Rational (рациональное число).

```

Сумма комплексных чисел 2 + 3i и 4 + 5i равна 6 + 8i
Разность комплексных чисел 2 + 3i и 4 + 5i равна -2 - 2i
Произведение комплексных чисел 2 + 3i и 4 + 5i равна -7 + 22i
Деление комплексных чисел 2 + 3i и 4 + 5i равна 0.5609756097560976 + 0.04878048780487805i
r1 = 3 / 4
r2 = 5 / 6
r1 + r2 = 19 / 12
r1 - r2 = -1 / 12
r1 * r2 = 5 / 8
r1 / r2 = 9 / 10
r3 = -5 / 3
r4 = -2 / 3
r3 + r4 = -7 / 3
r3 - r4 = -1
r3 * r4 = 10 / 9
r3 / r4 = 5 / 2
r5 = 2
r6 = 2
r5 + r6 = 4
r5 - r6 = 0
r5 * r6 = 4
r5 / r6 = 1
r5 == r6: True
r5 == r2: False

```

Рисунок 5. Результат работы программы индивидуального задания 2

```

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

```

```

from abc import ABC, abstractmethod
from math import gcd

```

```

class Pair(ABC):
    @abstractmethod
    def __add__(self, other):
        pass

    @abstractmethod

```

```

def __sub__(self, other):
    pass

@abstractmethod
def __mul__(self, other):
    pass

@abstractmethod
def __truediv__(self, other):
    pass

@abstractmethod
def __str__(self):
    pass

@property
@abstractmethod
def values(self):
    pass

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

class Complex(Pair):
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __add__(self, other):
        return Complex(self.real + other.real, self.imag + other.imag)

    def __sub__(self, other):
        return Complex(self.real - other.real, self.imag - other.imag)

    def __mul__(self, other):
        real_part = (
            self.real * other.real - self.imag * other.imag
        ) # Re(a * b)
        imag_part = (
            self.real * other.imag + self.imag * other.real
        ) # Im(a * b)
        return Complex(real_part, imag_part)

# Переопределяем деление для комплексных чисел
def __truediv__(self, other: Pair):
    denom = (
        other.real**2 + other.imag**2
    ) # Модуль другого числа в квадрате
    real_part = (
        self.real * other.real + self.imag * other.imag
    ) / denom # Re(a / b)
    imag_part = (
        self.imag * other.real - self.real * other.imag
    ) / denom # Im(a / b)
    return Complex(real_part, imag_part)

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

```

```
@property
def values(self):
    return self.real, self.imag
```

```
class Rational(Pair):
    def __init__(self, numerator: int, denominator: int = 1):
        if denominator == 0:
            raise ValueError("Недопустимое значение знаменателя")

        if not isinstance(numerator, int) or not isinstance(denominator, int):
            raise TypeError("Оба аргумента должны быть целыми числами!")

        self.numerator = numerator
        self.denominator = denominator

        self.integer_number = self.denominator == 1
        if not self.integer_number:
            self.__reduce()

    def __reduce(self):
        sign = 1
        if self.numerator * self.denominator < 0:
            sign = -1
        a, b = abs(self.numerator), abs(self.denominator)
        c = gcd(a, b)
        self.numerator = sign * (a // c)
        self.denominator = b // c
        self.integer_number = self.denominator == 1

    def __add__(self, other):
        x = (
            self.numerator * other.denominator
            + other.numerator * self.denominator
        )
        y = self.denominator * other.denominator
        r = Rational(x, y)
        r.__reduce()
        return r

    def __sub__(self, other):
        x = (
            self.numerator * other.denominator
            - other.numerator * self.denominator
        )
        y = self.denominator * other.denominator
        r = Rational(x, y)
        r.__reduce()
        return r

    def __mul__(self, other):
        x = self.numerator * other.numerator
        y = self.denominator * other.denominator
        r = Rational(x, y)
        r.__reduce()
        return r

    def __truediv__(self, other):
        x = self.numerator * other.denominator
        y = self.denominator * other.numerator
        r = Rational(x, y)
        r.__reduce()
        return r
```



```

def __str__(self):
    if self.integer_number:
        return f"{self.numerator}"
    return f"{self.numerator} / {self.denominator}"

@property
def values(self):
    return self.numerator, self.denominator

if __name__ == "__main__":
    first_complex = Complex(2, 3)
    second_complex = Complex(4, 5)
    print(
        f"Сумма комплексных чисел {first_complex} и {
            second_complex} равна {first_complex + second_complex}"
    )
    print(
        f"Разность комплексных чисел {first_complex} и {
            second_complex} равна {first_complex - second_complex}"
    )
    print(
        f"Произведение комплексных чисел {first_complex} и {
            second_complex} равна {first_complex * second_complex}"
    )
    print(
        f"Деление комплексных чисел {first_complex} и {
            second_complex} равна {first_complex / second_complex}"
    )

    r1 = Rational(3, 4)
    r2 = Rational(5, 6)
    print(f"r1 = {r1}")
    print(f"r2 = {r2}")
    print(f"r1 + r2 = {r1 + r2}")
    print(f"r1 - r2 = {r1 - r2}")
    print(f"r1 * r2 = {r1 * r2}")
    print(f"r1 / r2 = {r1 / r2}")

    r3 = Rational(5, -3)
    r4 = Rational(-6, 9)
    print(f"r3 = {r3}")
    print(f"r4 = {r4}")
    print(f"r3 + r4 = {r3 + r4}")
    print(f"r3 - r4 = {r3 - r4}")
    print(f"r3 * r4 = {r3 * r4}")
    print(f"r3 / r4 = {r3 / r4}")

    r5 = r6 = Rational(8, 4)
    print(f"r5 = {r5}")
    print(f"r6 = {r6}")
    print(f"r5 + r6 = {r5 + r6}")
    print(f"r5 - r6 = {r5 - r6}")
    print(f"r5 * r6 = {r5 * r6}")
    print(f"r5 / r6 = {r5 / r6}")

    print(f"r5 == r6: {r5 == r6}")
    print(f"r5 == r2: {r5 == r2}")

```

Ответы на контрольные вопросы:

1. Что такое наследование как оно реализовано в языке Python?

Наследование в Python позволяет классу-наследнику использовать атрибуты и методы родительского класса. Это обеспечивает повторное использование кода и расширение функциональности. Например, класс `Dog(Animal)` может наследовать метод `speak()` от класса `Animal`.

2. Что такое полиморфизм и как он реализован в языке Python?

Полиморфизм позволяет объектам разных классов иметь методы с одинаковыми именами, но с различной реализацией. В Python это реализуется через наследование и переопределение методов. Например, классы `Dog` и `Cat` могут иметь метод `speak()`, который выводит разные звуки для каждого животного.

3. Что такое "утиная" типизация в языке программирования Python?

"Утиная" типизация в Python заключается в том, что тип объекта определяется не его принадлежностью к определенному классу, а тем, имеет ли он необходимые методы и свойства. Если объект обладает требуемым поведением, он может использоваться в соответствующем контексте, независимо от его типа.

4. Каково назначение модуля `abc` языка программирования Python?

Модуль `abc` (Abstract Base Classes) предоставляет инструменты для создания абстрактных классов и методов. Абстрактные классы не могут быть инстанцированы напрямую и служат для того, чтобы обеспечить наличие определённых методов в подклассах.

5. Как сделать некоторый метод класса абстрактным?

Чтобы сделать метод класса абстрактным, используется декоратор `@abstractmethod` из модуля `abc`.

6. Как сделать некоторое свойство класса абстрактным?

Для создания абстрактного свойства класса применяется декоратор `@property` вместе с `@abstractmethod`.

7. Каково назначение функции `isinstance` ?

Функция `isinstance()` проверяет, является ли объект экземпляром указанного класса или его подкласса. Например, вызов `isinstance(dog, Animal)`

вернет True, если объект dog является экземпляром класса Animal или его наследников.

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