МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное

учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

«Наследование и полиморфизм в языке Python»

Отчет по лабораторной работе № 4.3

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-21-1

Гасанов Г. М. _____« » 2023г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2023

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

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия

MIT и язык программирования Python.

3. Выполните клонирование созданного репозитория.
4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. Проработайте примеры лабораторной работы.
8. Решите задачу:
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 -*-

"""
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.
"""
import random
  class Soldier:     def
__init__(self, number, team):
        self.number = number
self.team = team
     def follow_hero(self,
hero):
        print(f"Солдат {self.number} следует за героем {hero.number}.")
  class Hero:     def
__init__(self, number):
self.number = number
self.level = 1
     def
increase_level(self):
        self.level += 1
  if __name__ ==
'__main__':
    # Создаем героев для каждой команды
team1_hero = Hero(1)     team2_hero =
Hero(2)


    # Создаем списки солдат для каждой команды
team1_soldiers = []     team2_soldiers = []


    # Генерируем случайных солдат и определяем их принадлежность
команде     for i in range(10):         number = i + 1
        team = random.choice([team1_soldiers, team2_soldiers])
soldier = Soldier(number, team)         team.append(soldier)


    # Выводим длину списков солдат противоборствующих команд
print("Длина списка солдат команды 1:", len(team1_soldiers))
print("Длина списка солдат команды 2:", len(team2_soldiers))
```

```
    # Увеличиваем уровень героя команды с более длинным списком солдат
if len(team1_soldiers) > len(team2_soldiers):
team1_hero.increase_level()
        print("Уровень героя команды 1 увеличен.")
else:
        team2_hero.increase_level()
        print("Уровень героя команды 2 увеличен.")


    # Отправляем первого солдата первого героя следовать за ним
team1_soldiers[0].follow_hero(team1_hero)


    # Выводим идентификационные номера героя и солдата
print("Идентификационный номер героя:", team1_hero.number)
print("Идентификационный номер солдата:", team1_soldiers[0].number)
```

```
Длина списка солдат команды 1: 3
Длина списка солдат команды 2: 7
Уровень героя команды 2 увеличен.
Солдат 1 следует за героем 1.
Идентификационный номер героя: 1
Идентификационный номер солдата: 1

Process finished with exit code 0
```

Рисунок 1 – Результат работы программы

10. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

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

"""
Составить программу с использованием иерархии классов. Номер варианта
необходимо
получить у преподавателя. В раздел программы, начинающийся после инструкции if
__name__
= '__main__': добавить код, демонстрирующий возможности разработанных классов.


Создать класс Pair (пара чисел); определить методы изменения полей и
вычисления
произведения чисел. Определить производный класс RightAngled с полямикатетами.
Определить методы вычисления гипотенузы и площади треугольника.
"""
import math
  class Pair:     def
__init__(self, a, b):
        self.a = a
```

```
        self.b = b

    # Возможность изменить значения полей a и b объекта.
def set_values(self, a, b):
        self.a = a
self.b = b
    # Произведение
def multiply(self):
        return self.a * self.b


    # Сложение     def
calculate_numbers(self):
return self.a + self.b
  class RightAngled(Pair):
def __init__(self, a, b):
super().__init__(a, b)
     def set_values(self, a, b):
super().set_values(a, b)
     def
calculate_hypotenuse(self):
        return math.sqrt(self.a ** 2 + self.b ** 2)
     def
calculate_area(self):
        return 0.5 * self.a * self.b
  if __name__ ==
'__main__':
    # Создание объектов класса Pair и RightAngled
pair = Pair(3, 4)
    right_angled = RightAngled(3, 4)


    # Изменение значений полей и вычисление произведения чисел
pair.set_values(5, 6)
    print("Произведение чисел (Pair):", pair.multiply())
    # Изменение значений полей и вычисление суммы чисел
pair.set_values(7, 8)
    print("Сумма чисел (Pair):", pair.calculate_numbers())
    # Изменение значений полей и вычисление гипотенузы и площади треугольника
right_angled.set_values(5, 12)
    print("Гипотенуза (RightAngled):", right_angled.calculate_hypotenuse())
print("Площадь треугольника (RightAngled):", right_angled.calculate_area())
```

```
Произведение чисел (Pair): 30
Сумма чисел (Pair): 15
Гипотенуза (RightAngled): 13.0
Площадь треугольника (RightAngled): 30.0

Process finished with exit code 0
```

Рисунок 2 – Результат работы программы

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

"""
В следующих заданиях требуется реализовать абстрактный базовый класс,
определив в нем абстрактные методы и свойства. Эти методы определяются в
производных классах. В базовых
классах должны быть объявлены абстрактные методы ввода/вывода,
которые реализуются в производных классах.
Вызывающая программа должна продемонстрировать все варианты вызова
переопределенных
абстрактных методов. Написать функцию вывода, получающую параметры
базового класса по ссылке и демонстрирующую виртуальный вызов.


Создать абстрактный базовый класс Pair с виртуальными арифметическими
операциями.
Реализовать производные классы Complex (комплексное число) и Rational
(рациональное число).
"""
from abc import ABC, abstractmethod
  class Pair(ABC):
@abstractmethod     def
__add__(self, other):
        pass


    @abstractmethod     def
__sub__(self, other):
        pass


    @abstractmethod     def
__mul__(self, other):
        pass


    @abstractmethod
def __str__(self):
        pass


    @abstractmethod
def input(self):
        pass


    @abstractmethod
def output(self):
        pass
  class Complex(Pair):     def
__init__(self, real, imaginary):
        self.real = real
        self.imaginary = imaginary
     def __add__(self, other):
if isinstance(other, Complex):
            return Complex(self.real + other.real, self.imaginary +
other.imaginary)         else:
            raise TypeError("Unsupported operand type for +")
```

```
    def __sub__(self, other):
if isinstance(other, Complex):
            return Complex(self.real - other.real, self.imaginary -
other.imaginary)         else:
            raise TypeError("Unsupported operand type for -")
    def __mul__(self, other):
if isinstance(other, Complex):
            real_part = self.real * other.real - self.imaginary *
other.imaginary
            imaginary_part = self.real * other.imaginary + self.imaginary *
other.real
            return Complex(real_part, imaginary_part)
else:
            raise TypeError("Unsupported operand type for *")
    def __str__(self):
        return f"{self.real} + {self.imaginary}i"
    def input(self):
        real = float(input("Enter the real part: "))
imaginary = float(input("Enter the imaginary part: "))
self.real = real
        self.imaginary = imaginary
    def output(self):
print(self)
  class Rational(Pair):     def __init__(self,
numerator, denominator):
        self.numerator = numerator
self.denominator = denominator
    def __add__(self, other):
if isinstance(other, Rational):
            common_denominator = self.denominator * other.denominator
numerator = (self.numerator * other.denominator) +
(other.numerator * self.denominator)
            return Rational(numerator, common_denominator)
else:
            raise TypeError("Unsupported operand type for +")
    def __sub__(self, other):
if isinstance(other, Rational):
            common_denominator = self.denominator * other.denominator
numerator = (self.numerator * other.denominator) -
(other.numerator * self.denominator)
            return Rational(numerator, common_denominator)
else:
            raise TypeError("Unsupported operand type for -")
    def __mul__(self, other):
if isinstance(other, Rational):
            numerator = self.numerator * other.numerator
denominator = self.denominator * other.denominator
return Rational(numerator, denominator)         else:
            raise TypeError("Unsupported operand type for *")
    def __str__(self):
        return f"{self.numerator}/{self.denominator}"
```

```
    def
input(self):
        numerator = int(input("Enter the numerator: "))
denominator = int(input("Enter the denominator: "))
self.numerator = numerator          self.denominator =
denominator
    def
output(self):
print(self)
  def
demonstrate_virtual_call(pair):
    pair.output()
  if __name__ ==
'__main__':
    # Создание объектов класса Complex и Rational
complex1 = Complex(2, 3)      complex2 =
Complex(4, 5)      rational1 = Rational(1, 2)
rational2 = Rational(3, 4)


    # Продемонстрировать виртуальный вызов метода output
demonstrate_virtual_call(complex1)      demonstrate_virtual_call(rational1)


    # Примеры использования арифметических операций
result_complex = complex1 + complex2      print("Complex
addition:", result_complex)


    result_rational = rational1 - rational2
print("Rational subtraction:", result_rational)
```

```
2 + 3i
1/2
Complex addition: 6 + 8i
Rational subtraction: -2/8

Process finished with exit code 0
```

Рисунок 2 – Результат работы программы

11. Зафиксируйте сделанные изменения в репозитории.

12. Выполните слияние ветки для разработки с веткой main / master.

13. Отправьте сделанные изменения на сервер GitHub.

Контрольные вопросы:

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

Синтаксически создание класса с указанием его родителя выглядит так:

class имя_класса(имя_родителя1, [имя_родителя2,..., имя_родителя_n])

super – это ключевое слово, которое используется для обращения к родительскому классу.

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

Полиморфизм, как правило, используется с позиции переопределения методов базового класса в классе наследнике. Переопределение прописывается в классе-наследнике.

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

Утиная типизация – это концепция, характерная для языков программирования с динамической типизацией, согласно которой конкретный тип или класс объекта не важен, а важны лишь свойства и методы, которыми этот объект обладает. Другими словами, при работе с объектом его тип не проверяется, вместо этого проверяются свойства и методы этого объекта. Такой подход добавляет гибкости коду, позволяет полиморфно работать с объектами, которые никак не связаны друг с другом и могут быть объектами разных классов. Единственное условие, чтобы все эти объекты поддерживали необходимый набор свойств и методов.

4. Каково назназначение модуля abc языка программирования Python?

По умолчанию Python не предоставляет абстрактных классов. Python поставляется с модулем, который обеспечивает основу для определения абстрактных базовых классов (ABC), и имя этого модуля - ABC. ABC работает, декорируя методы базового класса как абстрактные, а затем регистрируя конкретные классы как реализации абстрактной базы.

5. Как сделать некоторый метод класса абстрактным?

Метод становится абстрактным, если он украшен ключевым словом @abstractmethod.

6. Как сделать некоторое свойство класса абстрактным?

Абстрактные классы включают в себя атрибуты в дополнение к методам, вы можете потребовать атрибуты в конкретных классах, определив их с помощью @abstractproperty.

7. Каково назначение функции isinstance ?

Встроенная функция isinstance(obj, Cls) , используемая при реализации методов арифметических операций и операций отношения, позволяет узнать что некоторый объект obj является либо экземпляром класса Cls либо экземпляром одного из потомков класса Cls.