

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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №26
дисциплины «Основы программной инженерии»

Выполнил:
Плугатырев Владислав Алексеевич
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и
сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

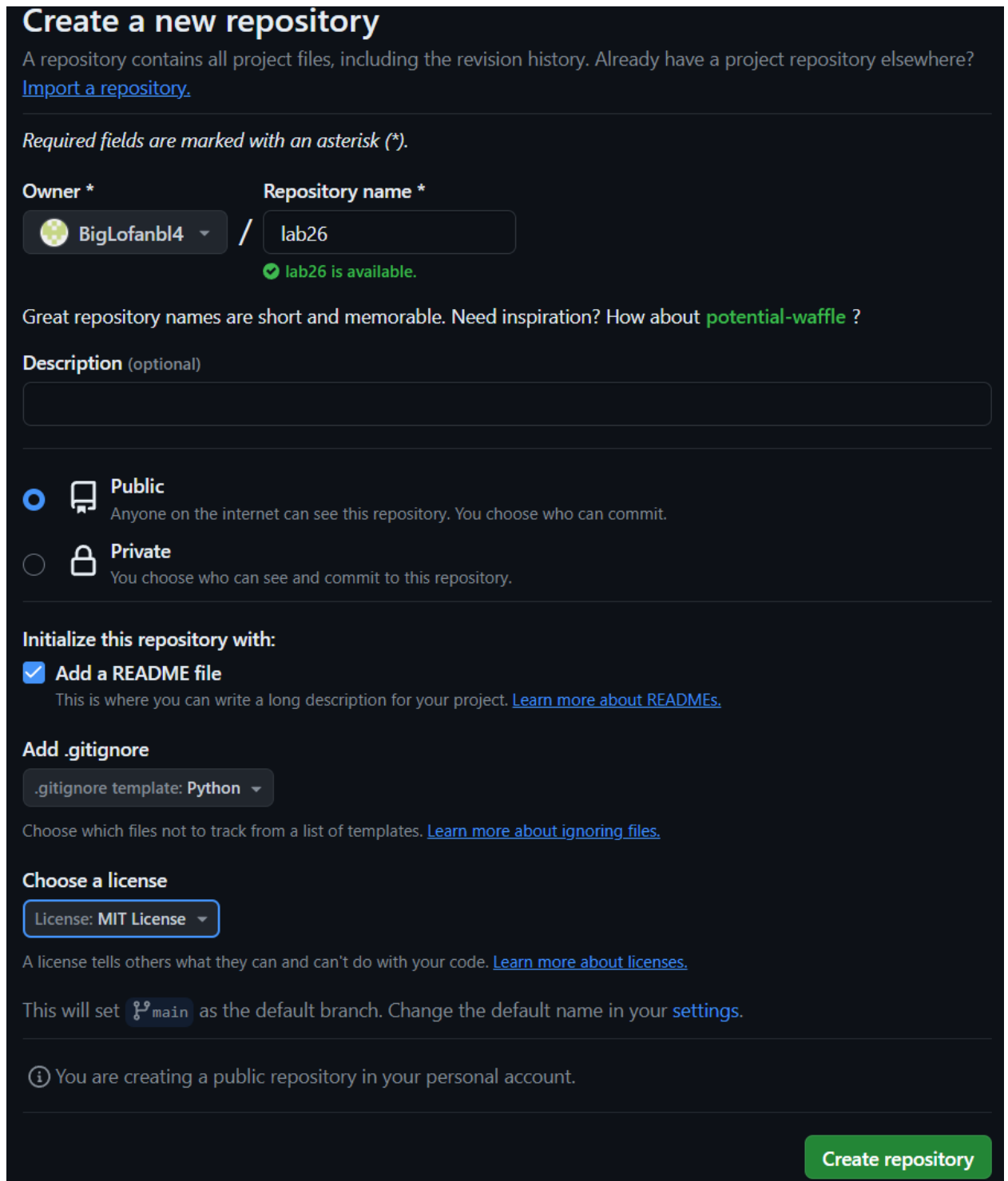
Ставрополь, 2024 г.

Тема: Наследование и полиморфизм в языке Python

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

Ход работы.

1. Создание репозитория.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk ().*

Owner * BigLofanbl4 / **Repository name *** lab26

✔ lab26 is available.

Great repository names are short and memorable. Need inspiration? How about **potential-waffle** ?

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

☒ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: MIT License

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set main as the default branch. Change the default name in your [settings](#).

(i) You are creating a public repository in your personal account.

Create repository

Рисунок 1.1 – Создание репозитория

2. Выполнение примеров из лабораторной работы.

```
class Rational:
    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)
        if b == 0:
            raise ValueError("Знаменатель не может быть нулем.")
        self.__numerator = a
        self.__denominator = b
        self.__reduce()

    # Сокращение дроби
    def __reduce(self):
        # Функция для нахождения наибольшего общего делителя
        def gcd(a, b):
            while b:
                a, b = b, a % b
            return a

        c = gcd(self.__numerator, self.__denominator)
        self.__numerator //= c
        self.__denominator //= c

    @property
    def numerator(self):
        return self.__numerator

    @property
    def denominator(self):
        return self.__denominator

    # Прочитать значение дроби с клавиатуры. Дробь вводится как a/b.
    def read(self, prompt=None):
        line = input(prompt) if prompt is not None else input()
        parts = list(map(int, line.split("/", maxsplit=1)))
        if parts[1] == 0:
            raise ValueError("Знаменатель не может быть нулем.")
        self.__numerator = parts[0]
        self.__denominator = parts[1]
        self.__reduce()
```

Рисунок 2.1 – Код первого примера

```
from abc import ABC, abstractmethod

class Polygon(ABC):
    @abstractmethod
    def noofsides(self):
        pass

class Triangle(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")

class Pentagon(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")

class Hexagon(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")

class Quadrilateral(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 4 sides")
```

Рисунок 2.2 – Код второго примера

```

class Animal(ABC):
    def move(self):
        pass

class Human(Animal):
    def move(self):
        print("I can walk and run")

class Snake(Animal):
    def move(self):
        print("I can crawl")

class Dog(Animal):
    def move(self):
        print("I can bark")

class Lion(Animal):
    def move(self):
        print("I can roar")

```

Рисунок 2.3 – Код третьего пример

3. Выполнение первого задания.

```

import random

class Human:
    def __init__(self, id, team):
        self.id = id
        self.team = team

class Soldier(Human):
    def __init__(self, id, team):
        super().__init__(id, team)

    def follow_hero(self, hero):
        print(f"Soldier {self.id} follows {hero.id} team {self.team}")

class Hero(Human):
    def __init__(self, id, team):
        super().__init__(id, team)
        self.level = 1

    def increase_level(self):
        self.level += 1

if __name__ == "__main__":
    red_hero = Hero("red_hero", "Blue")
    blue_hero = Hero("blue_hero", "Red")

    blue_soldiers = []
    red_soldiers = []

    for i in range(50):
        team = random.choice(["Blue", "Red"])
        if team == "Blue":
            blue_soldiers.append(Soldier(i, "Blue"))
        else:
            red_soldiers.append(Soldier(i, "Red"))

    print(f"Team blue has {len(blue_soldiers)} soldiers")
    print(f"Team red has {len(red_soldiers)} soldiers")

    if (len(blue_soldiers) > len(red_soldiers)):
        blue_hero.increase_level()
        blue_soldiers[0].follow_hero(blue_hero)
        print(blue_hero.id, blue_soldiers[0].id)
    else:
        red_hero.increase_level()
        red_soldiers[0].follow_hero(red_hero)
        print(red_hero.id, red_soldiers[0].id)

```

Рисунок 3.1 – Код задания

```

Team blue has 30 soldiers
Team red has 20 soldiers
Soldier 1 follows blue_hero team Blue
blue_hero 1

```

Рисунок 3.2 – Вывод задания

4. Выполнение первого индивидуального задания: создать класс Triad (тройка чисел); определить методы увеличения полей на 1. Определить производный класс Date с полями: год, месяц и день. Переопределить методы увеличения полей на 1 и определить метод увеличения даты на n дней.

```
from datetime import datetime, timedelta

class Triad:
    def __init__(self, num_1, num_2, num_3):
        self.num_1 = num_1
        self.num_2 = num_2
        self.num_3 = num_3

    def increment_num_1(self):
        self.num_1 += 1

    def increment_num_2(self):
        self.num_2 += 1

    def increment_num_3(self):
        self.num_3 += 1

    def __repr__(self):
        return f"({self.num_1}, {self.num_2}, {self.num_3})"

class Date(Triad):
    def __init__(self, year, month, day):
        super().__init__(year, month, day)

    # увеличение года на 1
    def increment_num_1(self):
        self.num_1 += 1

    # увеличение месяца на 1
    def increment_num_2(self):
        self.num_2 += 1
        if self.num_2 > 12:
            self.num_2 = 1
            self.increment_num_1()

    # увеличение дней на 1
    def increment_num_3(self):
        self.add_days(1)

    # увеличение на n дней
    def add_days(self, n):
        new_date = datetime(self.num_1, self.num_2, self.num_3) + timedelta(days=n)
        self.num_1 = new_date.year
        self.num_2 = new_date.month
        self.num_3 = new_date.day
```

Рисунок 4.1 – Код программы

```
(2022, 1, 11)
(2022, 1, 16)
```

Рисунок 4.2 – Вывод программы

5. Выполнение второго индивидуального задания: создать абстрактный базовый класс Series (прогрессия) с виртуальными функциями вычисления -го элемента прогрессии и суммы прогрессии. Определить производные классы: Linear (арифметическая) и Exponential (геометрическая).

```
from abc import ABC, abstractmethod

class Series(ABC):
    @abstractmethod
    def get_element(self, n):
        pass

    @abstractmethod
    def get_sum(self, n):
        pass

class Linear(Series):
    def __init__(self, a1, d):
        self.a1 = a1
        self.d = d

    def get_element(self, n):
        return self.a1 + (n - 1) * self.d

    def get_sum(self, n):
        return n * (self.a1 + self.get_element(n)) / 2

class Exponential(Series):
    def __init__(self, a1, q):
        self.a1 = a1
        self.q = q

    def get_element(self, n):
        return self.a1 * (self.q ** (n - 1))

    def get_sum(self, n):
        if self.q == 1:
            return n * self.a1
        else:
            return self.a1 * (1 - self.q ** n) / (1 - self.q)
```

Рисунок 5.1 – Код программы

```
16
235.0
54
59048.0
```

Рисунок 5.2 – Вывод программы

Ответы на контрольные вопросы

1. Наследование в Python: Это механизм, позволяющий новому классу (наследнику или подклассу) перенимать свойства и методы другого класса (родительского или суперкласса). Подклассы могут переопределять или расширять функциональность родительского класса. В Python наследование реализовано указанием родительского класса в скобках при определении нового класса.

2. Полиморфизм в Python: Это идиома, использующаяся для реализации функций или методов, способных обрабатывать данные различных типов. В Python полиморфизм проявляется в способности работать с объектами различных классов с использованием общего интерфейса. Полиморфизм реализуется за счет того, что функции и объекты не строго типизированы.

3. “Утиная” типизация (Duck Typing) в Python: Этот термин основан на фразе "если что-то ходит как утка и крякает как утка, то это, вероятно, утка". В контексте программирования это означает, что если объект реализует необходимый интерфейс (методы и свойства), то его можно использовать в качестве такового, независимо от того, каким классом он является. Python полагается на "утиную" типизацию, позволяя различным объектам использоваться в одном и том же контексте, если у них есть соответствующие методы и свойства.

4. Модуль abc (Abstract Base Classes) в Python: Этот модуль предоставляет инфраструктуру для определения абстрактных базовых классов, которые могут определять общие интерфейсы для набора подклассов. Это помогает в создании более надежного и легкого в использовании кода за счет предоставления классов-заготовок, которые требуют определенных методов у всех подклассов.

5. Создание абстрактного метода класса: Для создания абстрактного метода используются модуль «abc» и декоратор «abstractmethod». Абстрактный метод — это метод класса, который определен, но не реализован и должен быть переопределен в неабстрактных подклассах.

6. Создание абстрактного свойства класса: Механизм создания абстрактного свойства схож с созданием абстрактного метода, но используется декоратор «`abstractproperty`».

7. Назначение функции «`isinstance`»: Функция «`isinstance`» используется для проверки принадлежности объекта к классу или кортежу классов (для поддержки наследования). Это полезно, когда вы хотите проверить, является ли объект экземпляром определенного класса или его подкласса.