

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**
**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Кафедра инфокоммуникаций

Институт цифрового развития

ОТЧЁТ

по лабораторной работе

Дисциплина: «Объектно – ориентированное программирование»

Выполнил: студент 3 курса

группы ИВТ-б-о-21-1

Криворот Владимир Геннадьевич

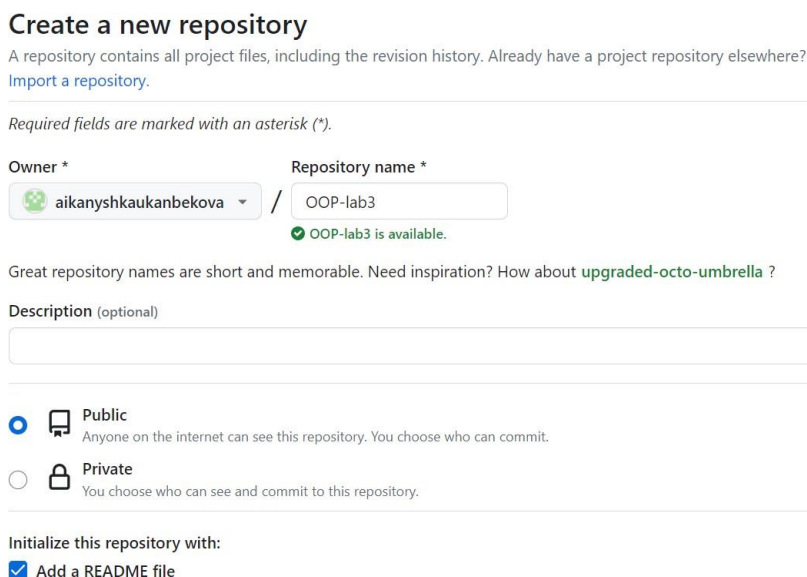
Ставрополь 2023

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.

Порядок выполнения работы:

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



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 * aikanyshkaukanbekova / Repository name * OOP-lab3

Great repository names are short and memorable. Need inspiration? How about [upgraded-octo-umbrella](#) ?

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

Рисунок 1. Создание репозитория

2. Выполните клонирование созданного репозитория.

```
C:\Users\User>cd C:\Users\User\Documents\3,1 курс\объектно-ориентированное\лаб 3
C:\Users\User\Documents\3,1 курс\объектно-ориентированное\лаб 3>git clone https://github.com/aikanyshkaukanbekova/OOP-lab3.git
Cloning into 'OOP-lab3'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.
C:\Users\User\Documents\3,1 курс\объектно-ориентированное\лаб 3>
```

Рисунок 2. Клонирование репозитория

3. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

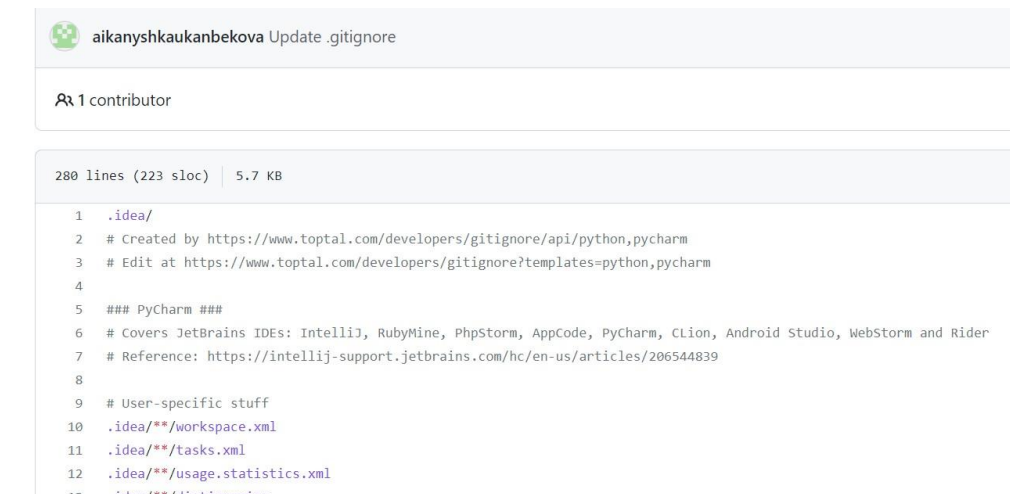


Рисунок 3. Дополнение файла .gitignore

Практическая часть:

Вариант 19

Задние 1. 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.

Код программы:

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-  
from abc import ABC  
import random
```

```

import sys

class GameUnit(ABC):
    """
    Базовый класс для игровых юнитов
    Имеет поля "Уникальный id" и "Принадлежность к команде"
    """
    def __init__(self, unique_id, team):
        self.unique_id = unique_id
        self.team = team

class Hero(GameUnit):
    """
    Класс "Герой", наследуется от игрового юнита
    Имеет поле "Уровень"
    """
    def __init__(self, unique_id, team):
        super().__init__(unique_id, team)
        self.level = 1

    def level_up(self):
        """
        Метод для повышения собственного уровня на 1
        """
        self.level += 1

class Soldier(GameUnit):
    """
    Класс "Солдат", наследуется от игрового юнита
    Имеет поле "Следует за героем" - в этом поле указывается объект "Герой",
    за которым следует текущий солдат
    """
    def __init__(self, unique_id, team):
        super().__init__(unique_id, team)
        self.follows_hero = None

    def follow(self, hero: Hero):
        """
        Метод для следования за героем
        """
        self.follows_hero = hero

class UniqueIdGenerator:
    """
    Класс для генерации последовательного id
    """
    def __init__(self):
        # Счетчик начинается с 0
        self.index = 0

    def generate_unique_id(self):
        # При каждой генерации нового числа - увеличиваем текущий счетчик на
        self.index = self.index + 1
        return self.index

if __name__ == '__main__':
    # Генератор уникальных id
    id_generator = UniqueIdGenerator()

```

```

# Герой красной команды
red_hero = Hero(id_generator.generate_unique_id(), "red")
# Герой синей команды
blue_hero = Hero(id_generator.generate_unique_id(), "blue")

soldiers_count = int(input("Введите общее количество солдат: "))
# Два массива для хранения солдат красной и синей команд
red_soldiers = []
blue_soldiers = []

# Генерируем героев
for x in range(soldiers_count):
    # Случайным образом определяем к какой команде будет причислен
    # созданный герой
    team = "red" if random.randint(1, 100) % 2 == 0 else "blue"
    new_soldier = Soldier(id_generator.generate_unique_id(), team)

    team_to_append = red_soldiers if team == "red" else blue_soldiers
    team_to_append.append(new_soldier)

if len(red_soldiers) == 0 and len(blue_soldiers) == 0:
    print("Ни одна из команд не содержит солдат")
    sys.exit()

# Определяем в какой команде большее количество солдат
[hero, soldiers] = [red_hero, red_soldiers] if len(red_soldiers) >
len(blue_soldiers) else [blue_hero,
blue_soldiers]

# Повышаем уровень этому герою
hero.level_up()
# Самый первый солдат теперь следует за своим героем
soldiers[0].follow(hero)

print(f"Герой: (unique_id: {hero.unique_id}, level: {hero.level}, team:
{hero.team})")
print(f"Идентификатор солдата, который следует за героем:
{soldiers[0].unique_id}")

```

```

Введите общее количество солдат: 20
Герой: (unique_id: 2, level: 2, team: blue)
Идентификатор солдата, который следует за героем: 4

```

Рисунок 1. Результат задания 1

Индивидуальные задания

Задание 1. 19 Создать базовый класс Triad (тройка чисел) с операциями сложения с числом, умножения на число, проверки на равенство. Создать производный класс Vector3D, задаваемый тройкой координат. Должны быть реализованы: операция сложения векторов, скалярное произведение векторов.

Код программы:

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

from typing import Self

class Triad:
    """
    Класс "Триада", содержит в себе 3 целочисленных значения
    """
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def add_number(self, number: int):
        """
        Операция сложения с числом
        """
        self.a = self.a + number
        self.b = self.b + number
        self.c = self.c + number

    def mul_number(self, number: int):
        """
        Операция умножения на число
        """
        self.a = self.a * number
        self.b = self.b * number
        self.c = self.c * number

    def __eq__(self, other: Self):
        """
        Операция сравнения с другой триадой
        """
        return self.a == other.a and self.b == other.b and self.c == other.c

    def display(self):
        """
        Печать на консоль
        """
        print(f"Triad(a: {self.a}, b: {self.b}, c: {self.c})")

class Vector3D(Triad):
    """
    Класс "Вектор", наследующийся от класса "Триада"
    """
    def add_vector(self, other: Self):
        """
        Определим метод сложения векторов
        """
        self.a = self.a + other.a
        self.b = self.b + other.b
        self.c = self.c + other.c

    def mul_vector(self, other: Self):
        """
        Определим метод умножения векторов
        """
        self.a = self.a * other.a
        self.b = self.b * other.b
```

```

        self.c = self.c * other.c

    def display(self):
        print(f"Vector3D(a: {self.a}, b: {self.b}, c: {self.c})")

if __name__ == '__main__':
    # Создаем две триады
    triad1 = Triad(10, 20, 30)
    triad2 = Triad(40, 80, 50)

    # К первой триаде добавим число 10
    triad1.add_number(10)
    triad1.display()

    # Умножим первую триаду на 2
    triad1.mul_number(2)
    triad1.display()

    # Сравним первую и вторую триаду
    print(triad1 == triad2)

    # Создадим два вектора
    vector1 = Vector3D(5, 7, 9)
    vector2 = Vector3D(2, 4, 6)

    # Добавим к первому вектору второй
    vector1.add_vector(vector2)
    vector1.display()

    # Умножим первый вектор на второй
    vector1.mul_vector(vector2)
    vector1.display()

```

```

Triad(a: 20, b: 30, c: 40)
Triad(a: 40, b: 60, c: 80)
False
Vector3D(a: 7, b: 11, c: 15)
Vector3D(a: 14, b: 44, c: 90)

```

Рисунок 2. Результат задания 2

Задание 2. Создать абстрактный базовый класс Number с абстрактными методами — арифметическими операциями. Создать производные классы Integer (целое) и Real (действительное).

Код программы:

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
from abc import ABC
from typing import Union, Self

```

```

class Number(ABC):
    """
    Абстрактный класс для числа
    """
    def __init__(self, value: Union[int, float]):
        self.value = value

    def add(self, other: Self):
        """
        Операция сложения
        """
        self.value = self.value + other.value

    def sub(self, other: Self):
        """
        Операция вычитания
        """
        self.value = self.value - other.value

    def mul(self, other: Self):
        """
        Операция умножения
        """
        self.value = self.value * other.value

    def div(self, other: Self):
        """
        Операция деления
        """
        self.value = self.value / other.value

    def __str__(self):
        return str(self.value)

class Integer(Number):
    """
    Класс "Целочисленное значение", наследуется от базового класса для всех чисел
    """
    def __init__(self, value: int):
        super().__init__(value)

    def add(self, other: Self):
        """
        Переопределенный метод сложения
        """
        self.value = int(self.value + other.value)

    def sub(self, other: Self):
        """
        Переопределенный метод вычитания
        """
        self.value = int(self.value - other.value)

    def mul(self, other: Self):
        """
        Переопределенный метод умножения
        """
        self.value = int(self.value * other.value)

    def div(self, other: Self):
        """

```



```

        Переопределенный метод деления
        """
        self.value = int(self.value / other.value)

class Real(Number):
    """
    Класс "Дробное число", наследуется от базового класса для всех чисел
    """
    def __init__(self, value: float):
        super().__init__(value)

if __name__ == "__main__":
    # Создадим число 10
    int1 = Integer(10)

    # Добавим к нему число 5
    int1.add(Integer(5))
    print(int1)

    # Разделим на 3
    int1.div(Integer(3))
    print(int1)

    # Умножим на 5
    int1.mul(Integer(5))
    print(int1)

    # Вычтем 10
    int1.sub(Integer(10))
    print(int1)

    # Создадим дробное число 10.5
    real1 = Real(10.5)

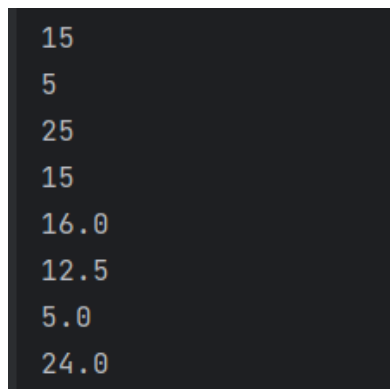
    # Добавим к нему другое дробное число 5.5
    real1.add(Real(5.5))
    print(real1)

    # Вычтем 3.5
    real1.sub(Real(3.5))
    print(real1)

    # Разделим на 2.5
    real1.div(Real(2.5))
    print(real1)

    # Умножим на 4.8
    real1.mul(Real(4.8))
    print(real1)

```



```
15
5
25
15
16.0
12.5
5.0
24.0
```

Рисунок 3. Результат задания 3

Контрольные вопросы:

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

Наследование в программировании - это механизм, который позволяет одному классу (подклассу) использовать свойства и методы другого класса (родительского класса). В Python наследование реализуется с помощью ключевого слова "class" и указания родительского класса в скобках после имени подкласса.

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

Полиморфизм в программировании означает способность объектов разных классов использовать одинаковые методы, но при этом вести себя по-разному. В Python полиморфизм реализуется благодаря динамической типизации и возможности переопределения методов.

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

"Утиная" типизация в языке программирования Python означает, что важнее не тип объекта, а его возможности и методы, которые он реализует. Это означает, что в Python мы можем вызывать методы объекта без явного указания его типа, и интерпретатор будет использовать методы, которые доступны у данного объекта.

Этот подход основан на принципе "если это выглядит как утка, плавает как утка и крикает как утка, то это, вероятно, и есть утка". То есть, важно не задавать объекту конкретный тип, а проверять его возможности и вызывать соответствующие методы.

4. Каково назначение модуля abc языка программирования Python?

Модуль abc (Abstract Base Classes) в языке программирования Python предназначен для создания абстрактных базовых классов. Абстрактный базовый класс (ABC) представляет собой класс, который может содержать абстрактные методы, то есть методы без реализации. Эти абстрактные методы должны быть переопределены в подклассах.

Назначение модуля abc в Python заключается в том, чтобы обеспечить стандартизацию интерфейсов для классов. Это позволяет создавать общие интерфейсы для различных классов, что упрощает использование и понимание кода.

5. Как сделать некоторый метод класса абстрактным?

Для того чтобы сделать метод класса абстрактным, необходимо использовать декоратор `@abstractmethod` из модуля abc.

6. Как сделать некоторое свойство класса абстрактным?

Для того чтобы сделать некоторое свойство класса абстрактным, можно воспользоваться аналогичным подходом с использованием декоратора `@abstractmethod`.

7. Каково назначение функции `isinstance` ?

Функция `isinstance()` в Python используется для проверки принадлежности объекта определенному классу или типу данных. Она принимает два аргумента: объект, который нужно проверить, и класс или тип данных, с которым нужно сравнить. Функция возвращает `True`, если объект принадлежит указанному классу или типу данных, и `False` в противном случае. Назначение функции `isinstance()` заключается в том, чтобы проверить тип объекта перед его использованием, что помогает избежать ошибок и неожиданного поведения программы.