МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Информатика»

Тема: Парадигмы программирования

Студентка гр. 3343 _____ Стрижков И.А.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург

2024

Цель работы

Цель данной лабораторной работы — изучить основы объектно-ориентированного программирования на примере Python. Основное внимание уделено работе с классами, созданию методов и функций для классов, пониманию принципов наследования, переопределения методов и работы с методом super().

Задание

Базовый класс - персонаж *Character*:

class:
Поля объекта класс Character:

- Пол (значение может быть одной из строк: 'm', 'w')
- Возраст (целое положительное число)
- Рост (в сантиметрах, целое положительное число)
- Вес (в кг, целое положительное число)
- При создании экземпляра класса Character необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

Воин - *Warrior*:

class Warrior: #Наследуется от класса Character
Поля объекта класс Warrior:

- Пол (значение может быть одной из строк: 'm', 'w')
- Возраст (целое положительное число)
- Рост (в сантиметрах, целое положительное число)
- Вес (в кг, целое положительное число)
- Запас сил (целое положительное число)
- Физический урон (целое положительное число)
- Количество брони (неотрицательное число)
- При создании экземпляра класса Warrior необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

В данном классе необходимо реализовать следующие методы:
Метод __str__():
Преобразование к строке вида: Warrior: Пол <пол>, возраст <возраст>, рост <рост>, вес <вес>, запас сил <запас сил>, физический урон <физический урон>, броня <количество брони>.
Метод __eq__():
Метод возвращает True, если два объекта класса равны и False иначе. Два объекта типа Warrior равны, если равны их урон, запас сил и броня.

Маг - *Magician*:

class Magician: #Наследуется от класса Character
Поля объекта класс Magician:

- Пол (значение может быть одной из строк: 'm', 'w')
- Возраст (целое положительное число)
- Рост (в сантиметрах, целое положительное число)
- Вес (в кг, целое положительное число)

- Запас маны (целое положительное число)
- Магический урон (целое положительное число)
- При создании экземпляра класса Magician необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

В данном классе необходимо реализовать следующие методы:
Метод __str__():
Преобразование к строке вида: Magician: Пол <пол>, возраст <возраст>, рост <рост>, вес <вес>, запас маны <запас маны>, магический урон <магический урон>.
Метод __damage__():
Метод возвращает значение магического урона, который может нанести маг, если потратит сразу весь запас маны (умножение магического урона на запас маны).

Лучник - *Archer*:

class Archer: #Наследуется от класса Character
Поля объекта класс Archer:

- Пол (значение может быть одной из строк: m (man), w(woman))
- Возраст (целое положительное число)
- Рост (в сантиметрах, целое положительное число)
- Вес (в кг, целое положительное число)
- Запас сил (целое положительное число)
- Физический урон (целое положительное число)
- Дальность атаки (целое положительное число)
- При создании экземпляра класса Archer необходимо убедиться, что переданные в конструктор параметры удовлетворяют требованиям, иначе выбросить исключение ValueError с текстом 'Invalid value'.

В данном классе необходимо реализовать следующие методы:
Метод __str__():
Преобразование к строке вида: Archer: Пол <пол>, возраст <возраст>, рост <рост>, вес <вес>, запас сил <запас сил>, физический урон <физический урон>, дальность атаки <дальность атаки>.
Метод __eq__():
Метод возвращает True, если два объекта класса равны и False иначе. Два объекта типа Archer равны, если равны их урон, запас сил и дальность атаки.
Необходимо определить список *list* для работы с персонажами:

Воины:

class WarriorList – список воинов - наследуется от класса list.
Конструктор:

1. Вызвать конструктор базового класса.

2. Передать в конструктор строку name и присвоить её полю name созданного объекта

Необходимо реализовать следующие методы:
Метод append(p_object): Переопределение метода append() списка. В случае, если p_object - Warrior, элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом: Invalid type <тип_объекта p_object>
Метод print_count(): Вывести количество воинов.

Маги:

class MagicianList – список магов - наследуется от класса list.
Конструктор:

1. Вызвать конструктор базового класса.
2. Передать в конструктор строку name и присвоить её полю name созданного объекта

Необходимо реализовать следующие методы:
Метод extend(iterable): Переопределение метода extend() списка. В случае, если элемент iterable - объект класса Magician, этот элемент добавляется в список, иначе не добавляется.
Метод print_damage(): Вывести общий урон всех магов.

Лучники:

class– список лучников - наследуется от класса list.
Конструктор:

1. Вызвать конструктор базового класса.
2. Передать в конструктор строку name и присвоить её полю name созданного объекта

Необходимо реализовать следующие методы:
Метод append(p_object): Переопределение метода append() списка. В случае, если p_object - Archer, элемент добавляется в список, иначе выбрасывается исключение TypeError с текстом: Invalid type <тип_объекта p_object>
Метод print_count(): Вывести количество лучников мужского пола.

Выполнение работы

В лабораторной работе необходимо создать классы с определёнными методами, которые представляют собой классы персонажа с определёнными параметрами и списки для хранения.

Класс Character является родительским для классов Warrior, Magician и Archer и хранит в себе информацию о пола, возраста, роста и веса объекта. При создании экземпляра класса проверяется, удовлетворяют ли переданные в конструктор параметры требованиям, иначе выводится исключение ValueError.

Класс Warrioror описывает война. Поля этого класса: пол, возраст, рост, вес, запас сил, физический урон, количество брони. Также реализованы метод, который выводит информацию об объекте и метод, который сравнивает два объекта этого класса по названию и автору.

Класс Magician описывает мага. Он содержит информацию об пол, возраст, рост, вес, запас маны, магический урон. Добавлены возвращает значение магического урона, который может нанести маг, если потратит сразу весь запас маны.

Класс Archer описывает мага. Он содержит информацию об пол, возраст, рост, вес, запас сил, физический урон, дальность атаки. Добавлены методы для вывода информации о лучнике и сравнения двух объектов класса, если равны название и страна — True, иначе — False.

Класс WarriorList — список войнов, наследуется от класса list. В классе переопределяется метод append() списка: если объект — воин, элемент добавляется в список, иначе исключение TypeError. Метод print_count() возвращает сумму всех имеющихся вйнов.

Класс MagicianList — список магов — наследуется от класса list. Переопределен метод extend() списка: если элемент iterable — объект класса Magician, этот элемент добавляется в список. Метод print_damage () выводит урон всех магов.

Класс ArcherList — список газет — наследуется от класса list. Переопределен метод append () списка: если элемент p_object — объект класса

Archer, этот элемент добавляется в список. Метод print_ print_count () выводит количество лучников мужского пола.

Метод __str__ является специальным методом, предназначенным для представления строкового представления объекта. Когда вызывается функция str() или встроенная функция print() для объекта, Python автоматически вызывает метод __str__, если он определен, чтобы получить строковое представление объекта.

Метод __eq__ в Python используется для определения логики сравнения двух объектов на равенство. Когда переопределяется метод __eq__ в классе, нужно определить, как объекты этого класса будут сравниваться при использовании оператора "==". Внутри метода __eq__ можно указать любую логику сравнения, которая необходима для структурного или значимого сравнения двух экземпляров класса.

При вызове выражения obj1 == obj2, Python автоматически вызывает метод __eq__ для объекта obj1 с передачей второго объекта obj2 в качестве аргумента. Метод __eq__ должен вернуть True, если объекты равны, и False, если они не равны.

Переопределенные методы append и extend в классе работают с помощью вызова super(). Это позволяет обращаться к методам append и extend из класса list и корректно добавлять объекты. Это сделано для того, чтобы гарантировать правильное поведение этих методов.

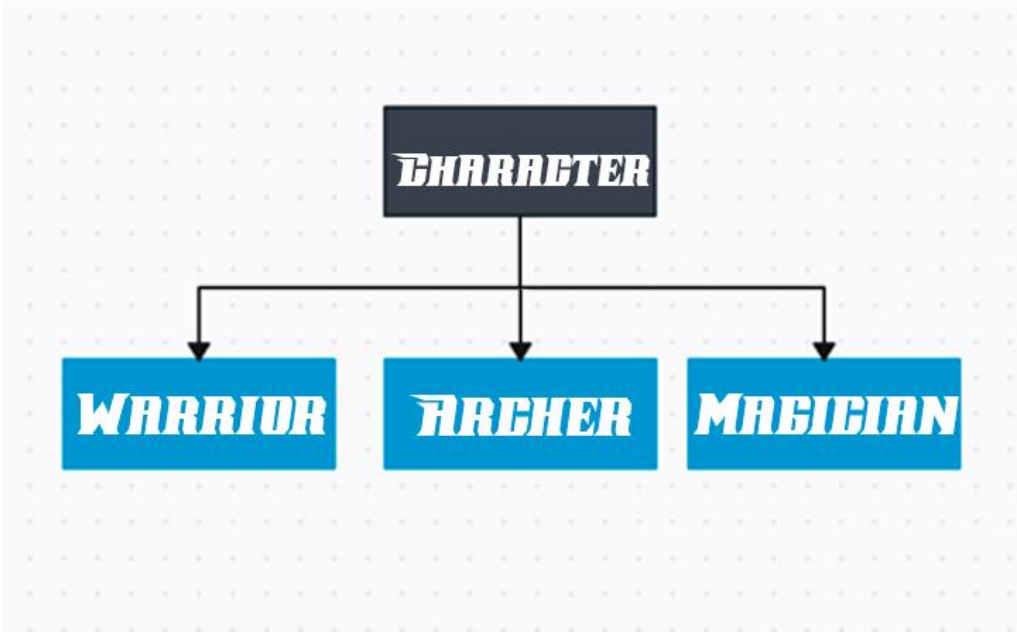

Рисунок 1 – Иерархия классов персонажей

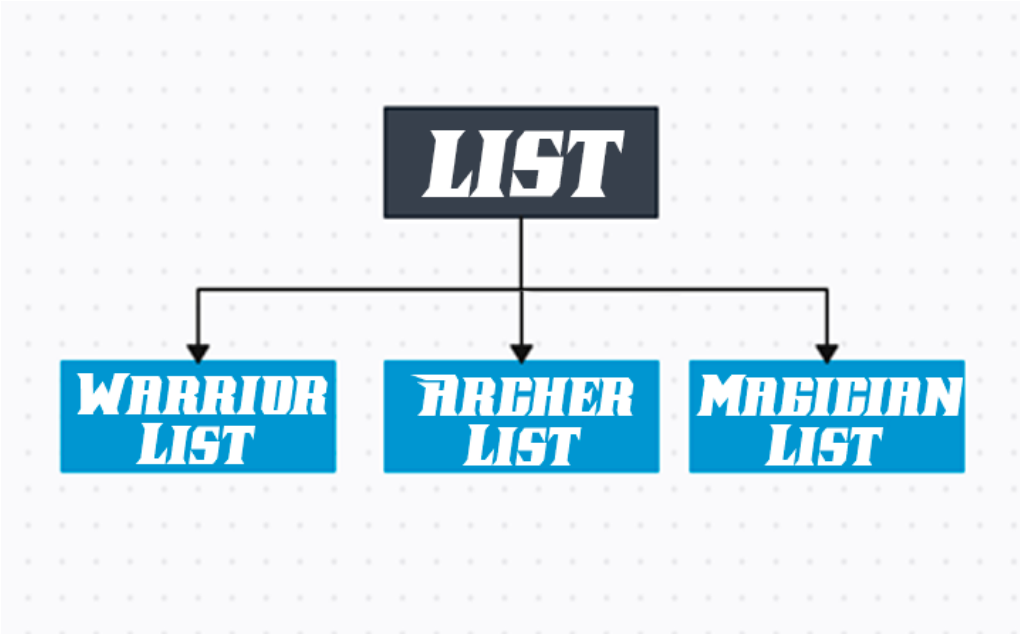

Рисунок 2 – Иерархия классов списков персонажей

Выводы

В ходе выполнения данной лабораторной работы были изучены основы объектно-ориентированного программирования на примере языка Python. Основной упор был сделан на работу с классами, создание методов и функций для классов, понимание принципов наследования, переопределения методов и использования метода super().

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Character:
    def __init__(self, gender, age, height, weight):
        if gender not in ['m', 'w'] or not (isinstance(age, int) and age > 0) or not (isinstance(height, int) and height > 0) or not (isinstance(weight, int) and weight > 0):
            raise ValueError('Invalid value')
        self.gender = gender
        self.age = age
        self.height = height
        self.weight = weight

class Warrior(Character):
    def __init__(self, gender, age, height, weight, forces, physical_damage, armor):
        super().__init__(gender, age, height, weight)
        if not (isinstance(forces, int) and forces > 0) or not (isinstance(physical_damage, int) and physical_damage > 0) or not (isinstance(armor, int) and armor > 0):
            raise ValueError('Invalid value')
        self.forces = forces
        self.physical_damage = physical_damage
        self.armor = armor

    def __str__(self):
        return f"Warrior: Пол {self.gender}, возраст {self.age}, рост {self.height}, вес {self.weight}, запас сил {self.forces}, физический урон {self.physical_damage}, броня {self.armor}."

    def __eq__(self, other):
        if not isinstance(other, Warrior):
            return False
        return self.physical_damage == other.physical_damage and self.forces == other.forces and self.armor == other.armor


class Magician(Character):
    def __init__(self, gender, age, height, weight, mana, magic_damage):
        super().__init__(gender, age, height, weight)
        if not (isinstance(mana, int) and mana > 0) or not (isinstance(magic_damage, int) and magic_damage > 0):
            raise ValueError('Invalid value')
        self.mana = mana
        self.magic_damage = magic_damage

    def __str__(self):
        return f"Magician: Пол {self.gender}, возраст {self.age}, рост {self.height}, вес {self.weight}, запас маны {self.mana}, магический урон {self.magic_damage}."

    def __damage__(self):
        return self.mana * self.magic_damage
```

```
class Archer(Character):
    def __init__(self, gender, age, height, weight, forces,
physical_damage, attack_range):
        super().__init__(gender, age, height, weight)
        if not (isinstance(forces, int) and forces > 0) or not
(isinstance(physical_damage, int) and physical_damage > 0) or not
(isinstance(attack_range, int) and attack_range > 0):
            raise ValueError('Invalid value')
        self.forces = forces
        self.physical_damage = physical_damage
        self.attack_range = attack_range

    def __str__(self):
        return f"Archer: Пол {self.gender}, возраст {self.age}, р
ост {self.height}, вес {self.weight}, запас сил {self.forces},
физический урон {self.physical_damage}, дальность атак
и {self.attack_range}."

    def __eq__(self, other):
        if not isinstance(other, Archer):
            return False
        return self.physical_damage == other.physical_damage and
self.forces == other.forces and self.attack_range == other.attack_range


class WarriorList(list):
    def __init__(self, name):
        super().__init__() # Вызываем конструктор базов
ого класса
        self.name = name

    def append(self, p_object):
        # Переопределение метода append()
        if not isinstance(p_object, Warrior):
            raise TypeError(f"Invalid type {type(p_object).__name__}")
        super().append(p_object)

    def print_count(self):
        # Используем метод count() для подсчета и выв
ода количества воинов
        print(f"{len(self)}")

class MagicianList(list):
    def __init__(self, name):
        super().__init__()
        self.name = name

    def extend(self, iterable):
        # Переопределение метода extend()
        for item in iterable:
            if isinstance(item, Magician):
                super().append(item)

    def print_damage(self):
```

```
        # Расчет и вывод общего урона всех магов
        total_damage = sum(magic.magic_damage for magic in self)
        print(f"{total_damage}")

class ArcherList(list):
    def __init__(self, name):
        super().__init__()
        self.name = name

    def append(self, p_object):
        # Переопределение метода append()
        if not isinstance(p_object, Archer):
            raise TypeError(f"Invalid type {type(p_object).__name__}")
        super().append(p_object)

    def print_count(self):
        # Подсчет и вывод количества лучников мужского пола
        male_archers_count = len([archer for archer in self if archer.gender == 'm'])
        print(f"{male_archers_count}")
```