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

# ОТЧЕТ

**по лабораторной работе №1**

**по дисциплине «Информационные технологии»**

**Тема:** Парадигмы программирования

Студент гр. 3344 _____ Пачев Д.К.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

**Цель работы**

Написать программу на языке Python, которая определяет набор классов для представления различных типов персонажей. Развить навыки написания кода с применением ООП.

**Задание**

**Вариант 2.**

**Базовый класс - персонаж *Character*:**

class Character:
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 ArcherList – 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**

1) Иерархия классов:
    - PositiveNumber
    - StringValue
    - Character
        - Warrior
        - Magician
        - Archer
    - list
        - WarriorList
        - MagicianList
        - ArcherList

PositiveNumber – класс, содержащий статический метод для проверки числовых значений на положительность и целочисленность. Реализует дескриптор для установки и получения целочисленных значений.

StringValue – класс, содержащий статический метод для проверки строковых значений на соответствие допустимым значениям. Реализует дескриптор для установки и получения строковых значений.

Character – базовый класс для персонажей, содержит атрибуты пола, возраста, роста и веса. Использует дескрипторы для установки и получения значений.

Warrior – класс-наследник от Character, представляющий воина. Содержит атрибуты силы, физического урона, брони и атрибуты родителя. Реализует методы для сравнения и вывода информации о воине.

Magician – класс-наследник от Character, представляющий мага. Содержит атрибуты маны, магического урона и атрибуты родителя. Реализует метод для расчета урона.

Archer – Класс-наследник от Character, представляющий лучника. Содержит атрибуты силы, физического урона, дальности атаки и атрибуты родителя. Реализует методы для сравнения и вывода информации о лучнике.

WarriorList, MagicianList, ArcherList – списочные классы, предназначенные для хранения объектов соответствующих классов, наследуются от класса list. Реализуют методы для добавления элементов, вывода информации.

2) Методы класса object, которые переопределены:

- __init__
- __str__
- __eq__

Дополнительные методы, которые переопределены:

- append в классе WarriorList и ArcherList
- extend в классе MagicianList

3) Метод __str__() будет использоваться при представлении объекта в виде строки, например при применения к объекту методов str() и print().

4) Переопределенные методы класса list для созданных списков будут работать, так как это такие же методы, только с дополнительной проверкой. Например, метод append класса ArcherList делает то же самое, что и стандартный метод append, просто проверяет перед этим, принадлежит ли объект классу Archer.

**Тестирование**

Результаты тестирования представлены в Таблице 1

Таблица 1 - Результаты тестирования

| № п/п | Входные данные | Выходные данные | Комментарии |
|---|---|---|---|
| 1. | character = Character('m', 20, 180, 70)<br>print(character.gender, character.age, character.height, character.weight)<br><br>warrior1 = Warrior('m', 20, 180, 70, 50, 100, 30)<br>warrior2 = Warrior('m', 20, 180, 70, 50, 100, 30)<br>print(warrior1.gender, warrior1.age, warrior1.height, warrior1.weight, warrior1.forces, warrior1.physical_damage, warrior1.armor)<br>print(warrior1.__str__())<br>print(warrior1.__eq__(warrior2)) | m 20 180 70<br>m 20 180 70 50 100 30<br>Warrior: Пол m, возраст 20, рост 180, вес 70, запас сил 50, физический урон 100, броня 30.<br>True | верно |
| 2. | warrior1 = Warrior('m', 20, 180, 70, 50, 100, 30)<br>warrior2 = Warrior('m', 20, 180, 70, 50, 100, 30)<br><br>warrior_list = WarriorList(Warrior)<br>warrior_list.append(warrior1)<br>warrior_list.append(warrior2)<br>warrior_list.print_count() | 2 | верно |

**Выводы**

В ходе выполнения лабораторной работы была разработана программа на языке Python для представления различных типов персонажей в виде классов, и развиты навыки написания кода, используя принципы ООП.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class PositiveNumber:
    @staticmethod
    def check_number(number):
        if not (isinstance(number, int) and number > 0):
            raise ValueError('Invalid value')

    def __set_name__(self, owner, name):
        self.name = '_' + name

    def __get__(self, instance, owner):
        return getattr(instance, self.name)

    def __set__(self, instance, value):
        self.check_number(value)
        setattr(instance, self.name, value)


class StringValue:
    @staticmethod
    def check_string(string):
        if not (isinstance(string, str) and string in 'mw'):
            raise ValueError('Invalid value')

    def __set_name__(self, owner, name):
        self.name = '_' + name

    def __get__(self, instance, owner):
        return getattr(instance, self.name)

    def __set__(self, instance, value):
        self.check_string(value)
        setattr(instance, self.name, value)


class Character:
    gender = StringValue()
    age = PositiveNumber()
    height = PositiveNumber()
    weight = PositiveNumber()

    def __init__(self, gender, age, height, weight):
        self.gender = gender
        self.age = age
        self.height = height
        self.weight = weight


class Warrior(Character):
```

```
    forces = PositiveNumber()
    physical_damage = PositiveNumber()
    armor = PositiveNumber()

    def __init__(self, gender, age, height, weight, forces,
physical_damage, armor):
        super().__init__(gender, age, height, weight)
        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 '
                f'{self.forces}, физический урон
{self.physical_damage}, броня {self.armor}.')

    def __eq__(self, other):
        return self.physical_damage == other.physical_damage and
self.forces == other.forces and self.armor == other.armor


class Magician(Character):
    mana = PositiveNumber()
    magic_damage = PositiveNumber()

    def __init__(self, gender, age, height, weight, mana,
magic_damage):
        super().__init__(gender, age, height, weight)
        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 '
                f'{self.mana}, '
                f'магический урон {self.magic_damage}.')

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


class Archer(Character):
    forces = PositiveNumber()
    physical_damage = PositiveNumber()
    attack_range = PositiveNumber()

    def __init__(self, gender, age, height, weight, forces,
physical_damage, attack_range):
        super().__init__(gender, age, height, weight)
        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}, рост
{self.height}, вес {self.weight}, запас сил {self.forces}, '
```

```
            f'физический урон {self.physical_damage}, дальность атаки
{self.attack_range}.')

    def __eq__(self, other):
        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__(self)
        self.name = name

    def append(self, p_object):
        if isinstance(p_object, Warrior):
            super().append(p_object)
        else:
            raise TypeError(f'Invalid type {type(p_object)}')

    def print_count(self):
        print(len(self))


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

    def extend(self, iterable):
        if all(isinstance(i, Magician) for i in iterable):
            super().extend(iterable)

    def print_damage(self):
        damage = 0
        for i in range(len(self)):
            damage += self[i].magic_damage
        print(damage)


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

    def append(self, p_object):
        if isinstance(p_object, Archer):
            super().append(p_object)
        else:
            raise TypeError(f'Invalid type {type(p_object)}')

    def print_count(self):
        count = 0
        for i in range(len(self)):
            if (self[i].gender == 'm'):
                count += 1
        print(count)
```