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.

Вариант 2

Студент гр. 3343 _____ Поддубный В. А

Преподаватель _____ Иванов Д. В.

Санкт-Петербург

2024

Цель работы

- Освоить принципы объектно-ориентированного программирования (ООП) в Python:
    - Работа с классами, создание методов и функций.
    - Наследование и переопределение методов.
    - Использование super().
- Создать программу, моделирующую работу с геометрическими фигурами.

Задание

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.

В отчете укажите:

1. Изображение иерархии описанных вами классов.

2. Методы, которые вы переопределили (в том числе методы класса object).

3. В каких случаях будут использованы методы __str__() и __print_damage__().

4. Будут ли работать переопределенные методы класса list для созданных списков? Объясните почему и приведите примеры.

Выполнение работы

Переопределенные методы:

- __init__(): В каждом классе переопределен для инициализации атрибутов объекта.
- __str__(): Переопределен в классах Warrior, Magician и Archer для предоставления строкового представления объектов.
- __eq__(): Переопределен в классах Warrior и Archer для сравнения объектов по определенным атрибутам.
- append(): Переопределен в классах WarriorList и ArcherList для добавления объектов только определенного типа.
- extend(): Переопределен в классе MagicianList для добавления объектов только определенного типа.

Использование методов __str__() и __print_damage__():

- **__str__():** Вызывается неявно при преобразовании объекта класса в строку, например, при использовании функции print() или при конкатенации со строкой.
- **__print_damage__():** Метод с таким именем в коде отсутствует. Вероятно, имелся в виду метод print_damage() класса MagicianList. Он вызывается явно для вывода суммарного магического урона всех магов в списке.

Работа переопределенных методов класса list:

Да, переопределенные методы класса list будут работать для созданных списков WarriorList, MagicianList и ArcherList, так как эти классы наследуют функциональность от list.

Переопределение методов append и extend позволяет контролировать типы объектов, добавляемых в списки, что обеспечивает безопасность типов.

Отработает корректно:

warriors.append(Warrior('m', 25, 180, 80, 100, 20, 50))

Вызовет TypeError:

warriors.append(Magician('w', 20, 165, 55, 150, 30))

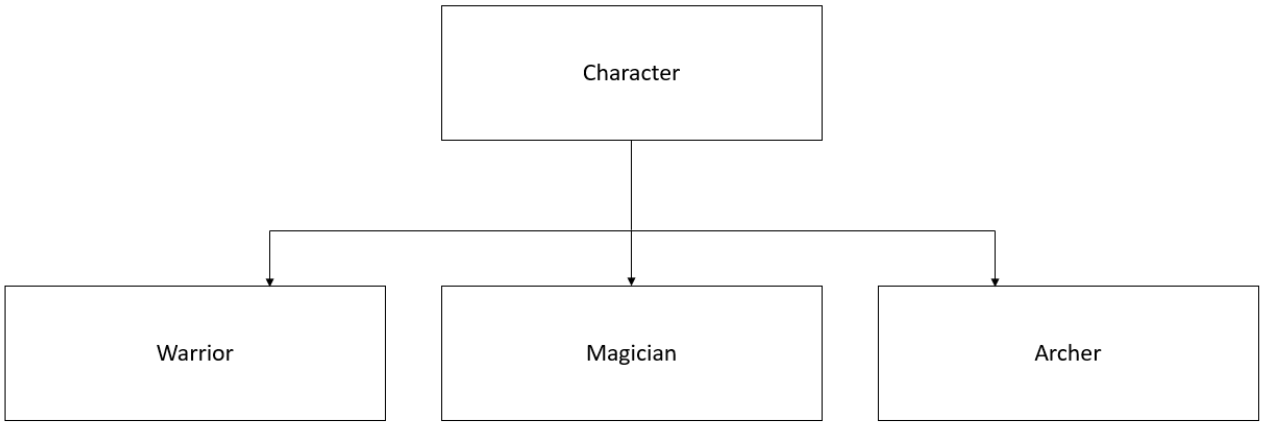

Рисунок 1 – Иерархия классов персонажей

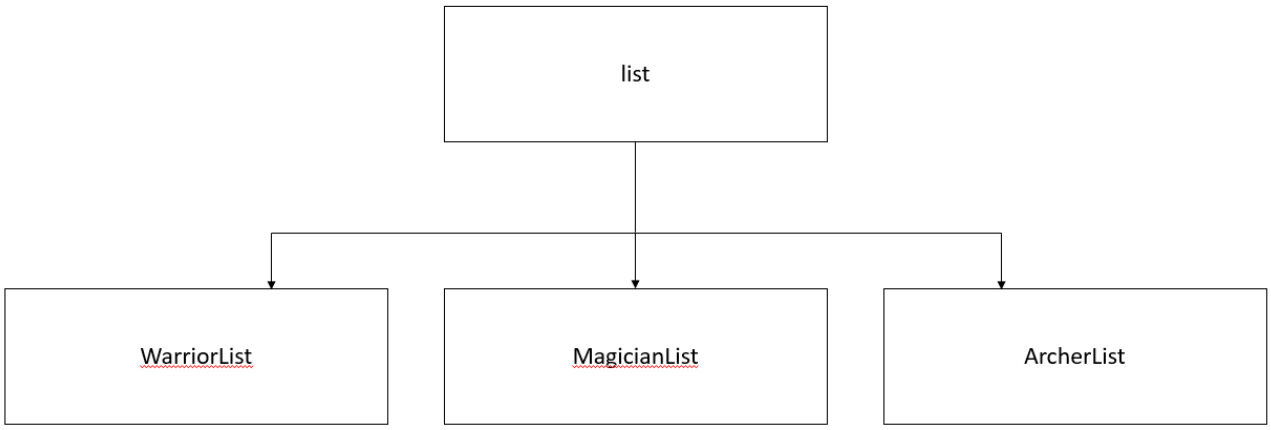

Рисунок 2 – Иерархия классов списков персонажей

Выводы

В рамках исследования были детально изучены следующие аспекты объектно-ориентированного программирования:

- **Механизм наследования**, позволяющий создавать новые классы на основе существующих, наследуя их атрибуты и методы. При этом допускается наследование как от одного, так и от нескольких родительских классов.
- **Переопределение методов**, дающее возможность изменять поведение унаследованных методов в дочерних классах, адаптируя их к специфике новых объектов.
- **Использование функции super()**, предоставляющей доступ к методам родительского класса из методов дочернего класса. Это особенно полезно при переопределении методов, когда требуется расширить функциональность родительского метода, сохраняя его базовую реализацию.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Character:

    def __init__(self, gender, age, height, weight):
        if gender not in ['m', 'w']:
            raise ValueError('Invalid value')
        if not isinstance(age, int) or age <= 0:
            raise ValueError('Invalid value')
        if not isinstance(height, int) or height <= 0:
            raise ValueError('Invalid value')
        if not isinstance(weight, int) or 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) or forces <= 0:
            raise ValueError('Invalid value')
        if not isinstance(physical_damage, int) or physical_damage <= 0:
            raise ValueError('Invalid value')
        if not isinstance(armor, int) or 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):
        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) or mana <= 0:
```

```
            raise ValueError('Invalid value')
        if not isinstance(magic_damage, int) or 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) or forces <= 0:
            raise ValueError('Invalid value')
        if not isinstance(physical_damage, int) or physical_damage <= 0:
            raise ValueError('Invalid value')
        if not isinstance(attack_range, int) or 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}, рост
{self.height}, вес {self.weight}, запас сил {self.forces}, 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.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).__name__}')

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

```
class MagicianList(list):

    def __init__(self, name):
        super().__init__()
        self.name = name

    def extend(self, iterable):
        for el in iterable:
            if isinstance(el, Magician):
                self.append(el)

    def print_damage(self):
        damage = 0
        for magician in self:
            damage += magician.magic_damage
        print(damage)


class ArcherList(list):

    def __init__(self, name):
        super().__init__()
        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).__name__}')

    def print_count(self):
        count = 0
        for archer in self:
            if archer.gender == 'm':
                count+=1
        print(count)
```