**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 Warriorоr 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.

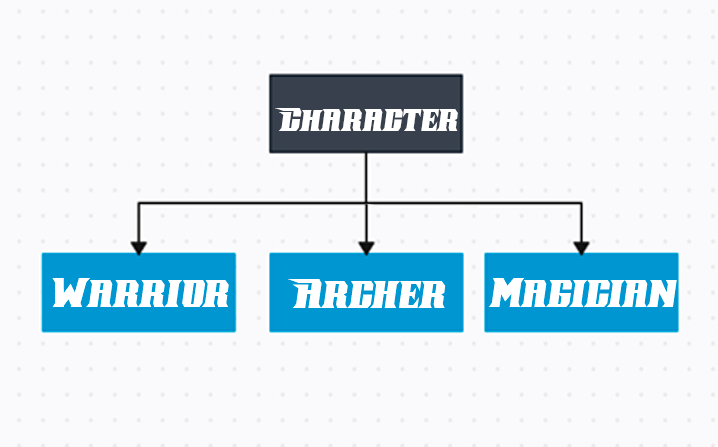


Рисунок 1 – Иерархия классов персонажей

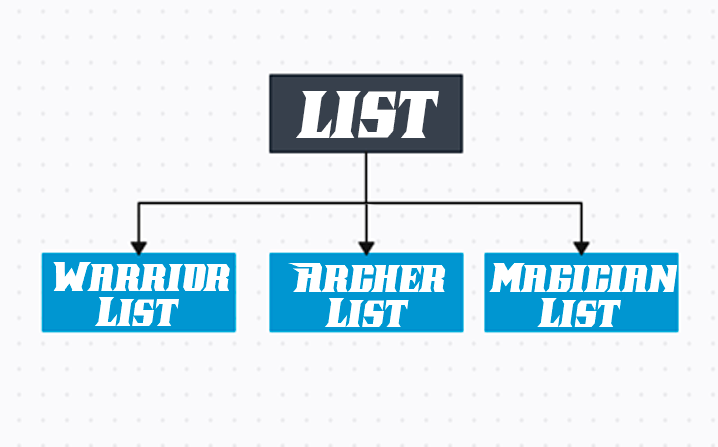


Рисунок 2 – Иерархия классов списков персонажей

## 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}, рост {self.height}, вес {self.weight}, запас сил {self.forces}, физический урон {self.physical\_damage}, дальность атаки {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 базового 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 вывода 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}")