**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 1

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Никишин С.А |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2024

## Цель работы

Научится работать с классами, создавать методы и функции для классов, понять принцип наследования и переопределения, понять, как работает super().

Программа должна быть разработана для создания и работы с различными классами, включая методы и функции для этих классов. Важно понимание наследования и переопределения в Python, а также использование `super()` для доступа к методам из родительского класса. Программа должна иметь возможность создания объектов различных фигур на основе различных классов и выполнения операций с ними. Она также должна проверять правильность типа данных и уметь добавлять объекты в определенную группу. Будет создана программа для работы с классами и объектами, включая методы, наследование, переопределение и проверку типов данных.

## 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**

В рамках лабораторной работы нужно реализовать классы, представляющие различные геометрические фигуры с заданными параметрами, а также создать списки для хранения этих фигур. Кроме того, необходимо определить методы для этих классов, которые будут выполнять определённые операции над фигурами.

В лабораторной работе предусмотрено создание класса Character, который будет являться родительским для классов Warrior, Magician и Archer. В этом классе будут содержаться такие характеристики, как пол, рост, возраст и вес. При инициализации экземпляра класса будет проводиться проверка входных данных на тип и корректность значений.

Метод \_\_str\_\_() используется для определения строкового представления объектов дочерних классов Warrior, Magician и Archer. Этот метод вызывается при преобразовании объекта класса в строковый тип (str).

Метод \_\_eq\_\_() отвечает за проверку на равенство дочерних классов Warrior и Archer.

Классы WarriorList, MagicianList и ArcherList наследуются от класса list и имеют переопределенный метод append для WarriorList и ArcherList. Этот метод работает так же, как и с любым другим списком, но при выполнении проверяется, принадлежит ли объект к соответствующему классу. Если объект не принадлежит к классу, то выбрасывается исключение TypeError.

Метод extend для класса MagicianList был переопределен таким образом, чтобы добавлять в конец списка только объекты, относящиеся к классу Magician. При этом все остальные объекты из итерируемого объекта игнорируются.

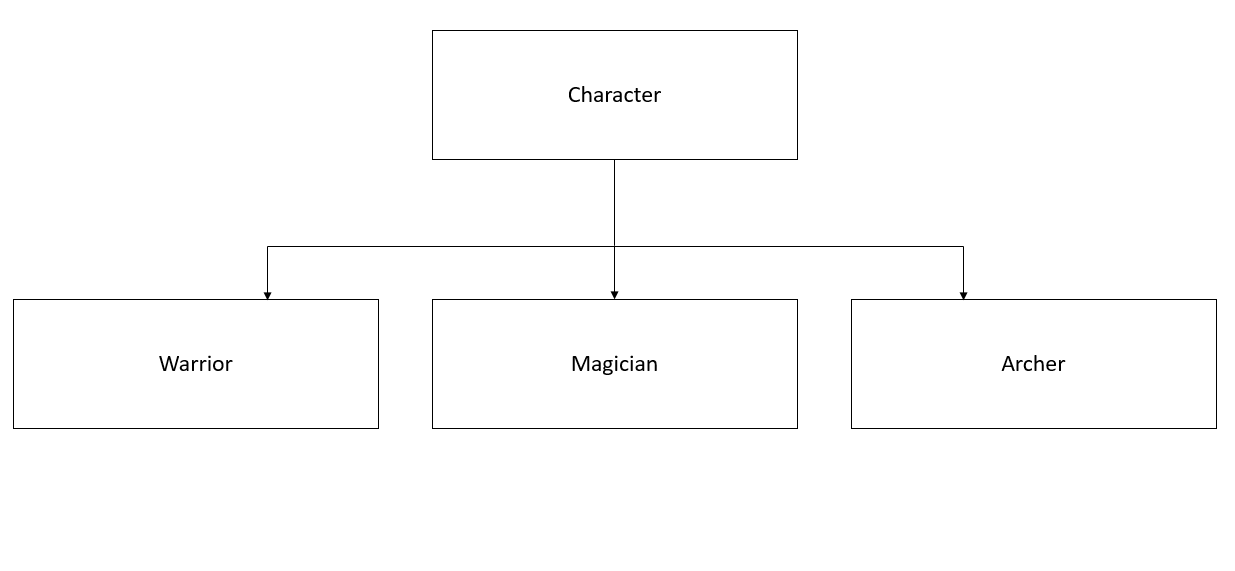


Рисунок 1 – Иерархия классов юнитов

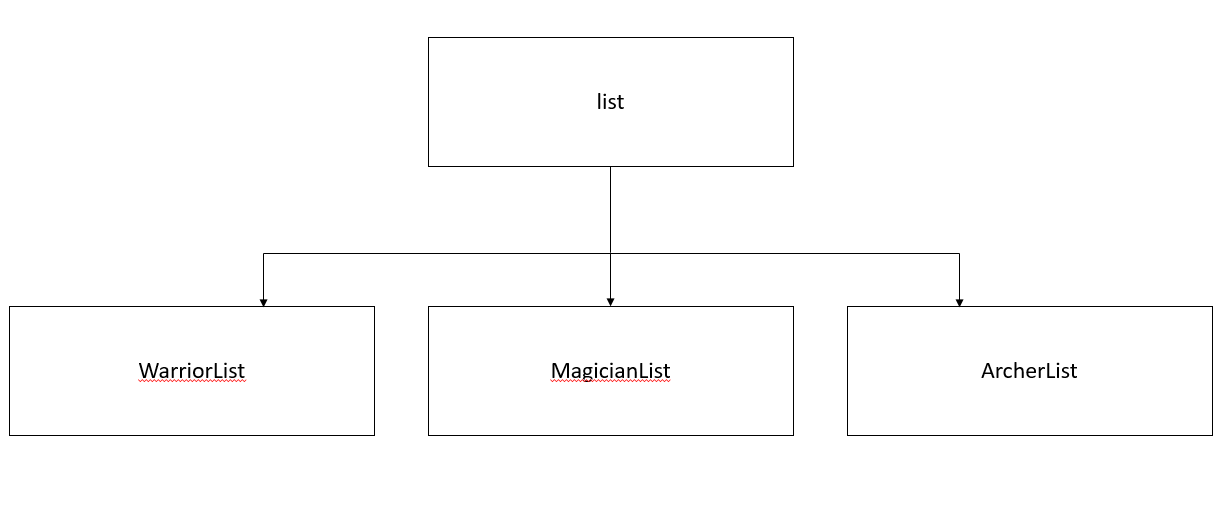


Рисунок 2 – Иерархия классов списков юнитов

## Выводы

В результате изучения наследования, переопределения методов и использования функции `super()`, была разработана программа, которая способна создавать объекты различных классов фигур, добавлять их в заданные группы и выполнять операции с ними.

# Приложение А Исходный код программы

Название файла: main.py

class Character:

def \_\_init\_\_(self,gender,age,height,weight):

if not(gender[0]=='m') and not(gender[0]=='w'):

raise ValueError ('Invalid value')

if not(isinstance(age,int)) or not(age>0):

raise ValueError ('Invalid value')

if not(isinstance(height,int)) or not(height>0):

raise ValueError ('Invalid value')

if not(isinstance(weight,int)) or not(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):

if not(isinstance(forces,int)) or not(forces>0):

raise ValueError ('Invalid value')

if not(isinstance(physical\_damage,int)) or not(physical\_damage>0):

raise ValueError ('Invalid value')

if not(isinstance(armor,int)) or not(armor>0):

raise ValueError ('Invalid value')

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 {self.forces}, физический урон {self.physical\_damage}, броня {self.armor}.'

def \_\_eq\_\_(self,other):

return True if (self.physical\_damage==other.physical\_damage) and (self.forces==other.forces) and (self.armor==other.armor) else False

class Magician(Character):

def \_\_init\_\_(self,gender,age,height,weight,mana,magic\_damage):

if not(isinstance(mana,int)) or not(mana>0):

raise ValueError ('Invalid value')

if not(isinstance(magic\_damage,int)) or not(magic\_damage>0):

raise ValueError ('Invalid value')

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 {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):

if not(isinstance(forces,int)) or not(forces>0):

raise ValueError ('Invalid value')

if not(isinstance(physical\_damage,int)) or not(physical\_damage>0):

raise ValueError ('Invalid value')

if not(isinstance(attack\_range,int)) or not(attack\_range>0):

raise ValueError ('Invalid value')

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}, физический урон {self.physical\_damage}, дальность атаки {self.attack\_range}.'

def \_\_eq\_\_(self,other):

return True if (self.physical\_damage==other.physical\_damage) and (self.forces==other.forces) and (self.attack\_range==other.attack\_range) else False

class WarriorList(list):

def \_\_init\_\_(self,name):

self.name=name

def append(self,p\_object):

if isinstance(p\_object,Warrior):

super().append(p\_object)

else:

raise TypeError ('Invalid type <тип\_объекта p\_object>')

def print\_count(self):

print (len(self))

class MagicianList(list):

def \_\_init\_\_(self,name):

self.name=name

def extend(self,iterable):

for iter in iterable:

if isinstance(iter,Magician):

super().append(iter)

def print\_damage(self):

print (sum([i.magic\_damage for i in list(self)]))

class ArcherList(list):

def \_\_init\_\_(self,name):

self.name=name

def append(self,p\_object):

if isinstance(p\_object,Archer):

super().append(p\_object)

else:

raise TypeError ('Invalid type <тип\_объекта p\_object>')

def print\_count(self):

print (len([i.gender for i in list(self) if i.gender[0]=='m']))