**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**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3388 |  | Кулач Д.В. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Изучение парадигм программирования, написание программы, используя подход ООП.

## Задание

Вариант 2

Базовый класс - персонаж *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 'I*nvalid 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:

Вызвать конструктор базового класса.

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:

Вызвать конструктор базового класса.

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:

Вызвать конструктор базового класса.

Передать в конструктор строку *name* и присвоить её полю *name* созданного объекта

Необходимо реализовать следующие методы:

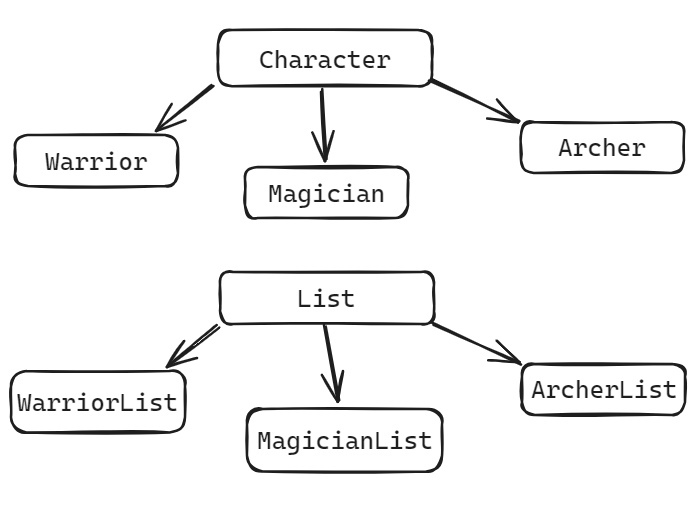
Метод *append(p\_object)*: Переопределение метода *append()* списка. В случае, если *p\_object* - *Archer*, элемент добавляется в список, иначе выбрасывается исключение *TypeError* с текстом:  *Invalid type* <тип\_объекта *p\_object*>

Метод *print\_count()*: Вывести количество лучников мужского пола.

## Выполнение работы

Порядок наследования изображен на рис. 1

Рисунок 1 — Иерархия классов



Переопределенные методы встроенного класса *object*:

* *\_\_init\_\_ -* инициализирует объект класса;
* *\_\_str\_\_* - возвращает строковое представление объекта;
* *\_\_eq\_\_* - позволяет проверить равенство объектов;

Переопределенные методы класса встроенного *list*:

* *append* — добавляет элемент в конец списка;
* *extend* — расширяет список, добавляя элементы из итерируемого объекта;

Метод *\_\_str\_\_* будет использован в следующих случаях:

* При вызове функции *print()* для объекта.
* При преобразовании объекта в строку с помощью функции *str()*.
* Если объект используется в форматированной строке.
* При использовании объекта в контексте, где требуется его строковое

представление, например, при выводе на консоль или записи в файл.

Во всех этих случаях *Python* будет пытаться вызвать метод *\_\_str\_\_*, если он определен для соответствующего класса объекта, чтобы получить строковое представление этого объекта.

Метод *\_\_print\_damage\_\_()* будет использован только при явном вызове метода — напрямую.

Будут ли работать переопределенные методы класса *list* для созданных списков?

В переопределенных методах *append()* и *extned()* добавляется проверка на класс объекта и вызывается соответствующий метод из родительского класса *list*, позволяя сохранять функциональность, предоставляемую родительским классом. Например, в классе *WarriorList* метод *append()* переопределен следующим образом: новый элемент будет добавлен в список только в том случае, если он является объектом класса *Warrior.*

Исходный код программы см. в приложении А.

## Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |

|  |  |  |  |
| --- | --- | --- | --- |
| 1. | character = Character('m', 20, 180, 70) #персонаж  print(character.gender, character.age, character.height, character.weight)  warrior1 = Warrior('m', 20, 180, 70, 50, 100, 30) #воин  warrior2 = Warrior('m', 20, 180, 70, 50, 100, 30)  print(warrior1.gender, warrior1.age, warrior1.height, warrior1.weight, warrior1.forces, warrior1.physical\_damage, warrior1.armor)  print(warrior1.\_\_str\_\_())  print(warrior1.\_\_eq\_\_(warrior2))  mag1 = Magician('m', 20, 180, 70, 60, 110) #маг  mag2 = Magician('m', 20, 180, 70, 60, 110)  print(mag1.gender, mag1.age, mag1.height, mag1.weight, mag1.mana, mag1.magic\_damage)  print(mag1.\_\_str\_\_())  print(mag1.\_\_damage\_\_())  archer1 = Archer('m', 20, 180, 70, 60, 95, 50) #лучник  archer2 = Archer('m', 20, 180, 70, 60, 95, 50)  print(archer1.gender, archer1.age, archer1.height, archer1.weight, archer1.forces, archer1.physical\_damage, archer1.attack\_range)  print(archer1.\_\_str\_\_())  print(archer1.\_\_eq\_\_(archer2))  warrior\_list = WarriorList(Warrior) #список воинов  warrior\_list.append(warrior1)  warrior\_list.append(warrior2)  warrior\_list.print\_count()  mag\_list = MagicianList(Magician) #список магов  mag\_list.extend([mag1, mag2])  mag\_list.print\_damage()  archer\_list = ArcherList(Archer) #список лучников  archer\_list.append(archer1)  archer\_list.append(archer2)  archer\_list.print\_count() | m 20 180 70  m 20 180 70 50 100 30  Warrior: Пол m, возраст 20, рост 180, вес 70, запас сил 50, физический урон 100, броня 30.  True  m 20 180 70 60 110  Magician: Пол m, возраст 20, рост 180, вес 70, запас маны 60, магический урон 110.  6600  m 20 180 70 60 95 50  Archer: Пол m, возраст 20, рост 180, вес 70, запас сил 60, физический урон 95, дальность атаки 50.  True  2  220  2 | OK |
| ... |  |  |  |

## Выводы

Были изучены парадигмы программирования и получены навыки работы с классами в Python. В результате была написана программа, использующая принципы ООП с помощью классов Python.

# Приложение А Исходный код программы

Название файла: main.py

class Character:

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

if not((gender == 'm' or gender == 'w') and isinstance(age, int) and age > 0 and isinstance(height, int) and height > 0 and

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 and isinstance(physical\_damage, int) and physical\_damage > 0 and

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):

return self.forces == other.forces and self.armor == other.armor and self.physical\_damage == other.physical\_damage

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 and 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.magic\_damage\*self.mana

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 and isinstance(physical\_damage, int) and physical\_damage > 0

and 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):

return self.forces == other.forces and self.attack\_range == other.attack\_range and self.physical\_damage == other.physical\_damage

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 <тип\_объекта p\_object>'))

def print\_count(self):

print(len(self))

class MagicianList(list):

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

super().\_\_init\_\_()

self.name = name

def extend(self, iterable):

super().extend(item for item in iterable if isinstance(item, Magician))

def print\_damage(self):

print(sum(item.magic\_damage for item in self))

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 <тип\_объекта p\_object>'))

def print\_count(self):

print(sum(1 for item in self if item.gender == 'm'))