**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. Вариант 2.

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Калиберов Н.И |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Рассмотреть парадигмы программирования на примере принципов наследования классов и переопределения методов из объектно-ориентированного программирования.

## Задание

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.

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 классе Warrior были переопределены методы \_\_str\_\_, отвечающий за строковое представление имени объекта и всех его характеристик, и \_\_eq\_\_, отвечающее за сравнение с другими объектами этого же класса, вызван метод super().\_\_init\_\_().

В классе Magician были переопределены методы \_\_str\_\_, отвечающий за строковое представление имени объекта и всех его характеристик, и \_\_damage\_\_, подсчитывающий общий урон всех магов, вызван метод super().\_\_init\_\_().

В классе Archer были переопределены методы \_\_str\_\_, отвечающий за строковое представление имени объекта и всех его характеристик, и \_\_eq\_\_, отвечающее за сравнение с другими объектами этого же класса, вызван метод super().\_\_init\_\_().

В классе WarriorList был переопределен метод append(p\_object), добавляющий элемент в конец списка, если он соответствует этому классу, иначе выводит соответствующую ошибку и определен метод print\_count, который выводит количество имеющихся войнов.

В классе MagicianList был переопределен метод \_\_extend\_\_(iterable), добавляющий элемент в конец списка, если он соответствует этому классу, иначе выводит соответствующую ошибку и определен метод print\_damage, который выводит общий магический урон.

В классе ArcherList был переопределен метод append(p\_object), добавляющий элемент в конец списка, если он соответствует этому классу, иначе выводит соответствующую ошибку и определен метод print\_count, который выводит количество лучников мужского пола.

## Тестирование

Здесь результаты тестирования, которые помещаются на одну страницу.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 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: Pm, возраст 20, рост 180, вес 70, запас сил 50, физический урон 100, броня 30.  True  m 20 180 70 60 110  Magician: P m, возраст 20, рост 180, вес 70, запас маны 60, магический урон 110.  6600  m 20 180 70 60 95 50  Archer: W m, возраст 20, рост 180, вес 70, запас сил 60, физический урон 95, дальность атаки 50.  True  2  220  2 | Ответ верный |

## Выводы

Были изучены принципы объектно-ориентированного программирования. Закреплены навыки создания классов, их наследников, создания и переопределения методов. Была построена иерархия классов и созданы методы для работы с ними.

# 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') or not all(isinstance(x, int) and x > 0 for x in [age, height, weight]):

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(gender == 'm' or gender == 'w') or not all(isinstance(x, int) and x > 0 for x in [age, height, weight, forces, physical\_damage]) or not (armor > 0 and isinstance(armor, (int, float))):

raise ValueError('Invalid value')

else:

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, another\_player):

return isinstance(another\_player, type(self))

class Magician(Character):

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

if not(gender == 'm' or gender == 'w') or not all(isinstance(x, int) and x > 0 for x in [age, height, weight, mana, magic\_damage]):

raise ValueError('Invalid value')

else:

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(gender == 'm' or gender == 'w') or not all(isinstance(x, int) and x > 0 for x in [age, height, weight, forces, physical\_damage, attack\_range]):

raise ValueError('Invalid value')

else:

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, another\_player):

return isinstance(another\_player, type(self))

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(type(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):

for i in range(len(iterable)):

if isinstance(iterable[i], Magician):

super().append(iterable[i])

def print\_damage(self):

print(sum(self[x].magic\_damage for x in range(len(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(type(p\_object))

def print\_count(self):

print(len(self))