**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Информатика»**

Тема: Парадигмы программирования

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Костромитин М.М. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Целью работы является написание программы на языке Python, в которой будут реализованы классы и их иерархия, а также написаны классы, которые наследуются от стандартных классов языка Python. В процессе написания программы необходимо изучить принципы объектно-ориентированного программирования.

## Задание

### Вариант 2

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:

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

Ниже представлены шаги выполнения лабораторной работы:

1. Создание класса Character, который содержит поля gender, age, height, weight, в его конструкторе реализованная проверка правильности ввода значений полей, в случае неверного ввода выкидывается ошибка ValueError.
2. Создание класса Warrior, который наследуется от Character, в его конструкторе определены дополнительные поля forces, physical\_damage, armor, так же как и в классе Character их значения проверяются и в случае неверного ввода выкидывается ошибка, также переопределенны методы класса object: \_\_str\_\_ и \_\_eq\_\_.
3. Создание класса Magician, который наследуется от Character, в его конструкторе определены дополнительные поля mana, magic\_damage, так же как и в классе Character их значения проверяются и в случае неверного ввода выкидывается ошибка, также переопределенны методы класса object: \_\_str\_\_, и написан метод \_\_damage\_\_.
4. Создание класса Archer, который наследуется от Character, в его конструкторе определены дополнительные поля forces, physical\_damage, attack\_range, так же как и в классе Character их значения проверяются и в случае неверного ввода выкидывается ошибка, также переопределенны методы класса object: \_\_str\_\_ и \_\_eq\_\_.
5. Создание класса WarriorList, который наследуется от класса list, в нем переопределен метод класса list: append – в лист могут быть добавлены только представители класса Warrior, а также определен метод print\_count, который выводит количество элементов листа.
6. Создание класса MagicianList, который наследуется от класса list, в нем переопределен метод класса list: extend – в лист могут быть добавлены только представители класса Magician, а также определен метод print\_damage, который выводит сумму поля magic\_damage у всех элементов листа.
7. Создание класса ArcherList, который наследуется от класса list, в нем переопределен метод класса list: append – в лист могут быть добавлены только представители класса Archer, а также определен метод print\_count, который выводит количество элементво листа, у которых поле gender равен m.

Иерархию описанных классов можно изобразить так:

1. object
   1. Character
      1. Warrior
      2. Magician
      3. Archer
   2. list
      1. WarriorList
      2. MagicianList
      3. ArcherList

Список переопределенных методов внутри классов:

1. \_\_init\_\_ - переопределен во всех классах для определения их полей.
2. \_\_str\_\_ - переопределен в класса Warrior, Magician, Archer для удобного представления информации о представителе класса.
3. \_\_eq\_\_ - переопределен в классах Warrior, Archer для проверки на равенство двух представителей классов, сравниваются их поля: для Warrior – physical\_damage, forces, armor; для Archer – forces, physical\_damage, attack\_range.
4. append, extend – переопределены для классов WarriorList, MagicianList, ArcherList, для того, чтобы в классы могли быть добавлены только представители классов Warrior, Magician, Archer соответственно.

Метод \_\_str\_\_() будет использован тогда, когда необходимо строковое представления представителя класса Warrior, Magician, Archer. Метод print\_damage() будет использован, когда необходимо узнать сумму полей magic\_damage у представителей класса Magician внутри класса MagicianList.

Переопределенные методы класса list для созданных списков будут работать, потому что они написаны без использования методов класса list, и не зависят от него, пример:

listik = WarriorList('Name')

listik.append(Warrior('m', 30, 50, 20, 100, 140, 23))

print(len(listik)) # выведется 1

## Тестирование

Результаты тестирования представлены в Таблице 1.

Таблица 1.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1 | try: #неправильные данные для воина  warrior1 = Warrior(-1, 20, 180, 70, 50, 100, 30)  except (TypeError, ValueError):  print('OK')  try:  warrior1 = Warrior('m', -1, 180, 70, 50, 100, 30)  except (TypeError, ValueError):  print('OK')  try: #неправильные данные для мага  mag1 = Magician(-1, 20, 180, 70, 60, 110)  except (TypeError, ValueError):  print('OK')  try:  mag1 = Magician('m', -1, 180, 70, 60, 110)  except (TypeError, ValueError):  print('OK')  try: #неправильные данные для лучника  archer1 = Archer(-1, 20, 180, 70, 60, 95, 50)  except (TypeError, ValueError):  print('OK')  try:  archer1 = Archer('m', -1, 180, 70, 60, 95, 50)  except (TypeError, ValueError):  print('OK') | OK  OK  OK  OK  OK  OK | Неправильные данные для инициализации представителей классов |
| 2 | 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 | Проверка всех написанных классов |

## Выводы

Была написана программа, содержащая иерархию классов, для которых написаны свои методы и переопределенные методы старших по иерархии классов. Была изучена объектно-ориентированная парадигма программирования.

# Приложение А Исходный код программы

Исходный файл: main.py

class Character:

try:

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

if gender not in ['m', 'w'] or not (type(age) == int and age > 0) or not (

type(height) == int and height > 0) or not (type(weight) == int and weight > 0):

raise ValueError

self.gender = gender

self.age = age

self.height = height

self.weight = weight

except ValueError:

print("Invalid value")

class Warrior(Character):

try:

def \_\_init\_\_(self, gender, age, height, weight, forces, physical\_damage, armor):

super().\_\_init\_\_(gender, age, height, weight)

if not (type(forces) == int and forces > 0) or not (

type(physical\_damage) == int and physical\_damage > 0) or not (type(armor) == int and armor > 0):

raise ValueError

self.forces = forces

self.physical\_damage = physical\_damage

self.armor = armor

except ValueError:

print("Invalid value")

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

return True

return False

class Magician(Character):

try:

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

super().\_\_init\_\_(gender, age, height, weight)

if not (type(mana) == int and mana > 0) or not (type(magic\_damage) == int and magic\_damage > 0):

raise ValueError

self.mana = mana

self.magic\_damage = magic\_damage

except ValueError:

print("Invalid value")

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):

try:

def \_\_init\_\_(self, gender, age, height, weight, forces, physical\_damage, attack\_range):

super().\_\_init\_\_(gender, age, height, weight)

if not (type(forces) == int and forces > 0) or not (

type(physical\_damage) == int and physical\_damage > 0) or not (

type(attack\_range) == int and attack\_range > 0):

raise ValueError

self.forces = forces

self.physical\_damage = physical\_damage

self.attack\_range = attack\_range

except ValueError:

print("Invalid values")

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

return True

return False

class WarriorList(list):

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

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

self.name = name

try:

def append(self, p\_object):

if type(p\_object) == Warrior:

self[len(self):] = [p\_object]

else:

raise TypeError(type(p\_object))

except TypeError as Err:

print(f'Invalid type {Err}')

def print\_count(self):

print(len(self))

class MagicianList(list):

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

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

self.name = name

def extend(self, iterable):

for value in iterable:

if type(value) == Magician:

self[len(self):] = [value]

def print\_damage(self):

result = 0

for value in self:

result += value.magic\_damage

print(result)

class ArcherList(list):

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

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

self.name = name

try:

def append(self, p\_object):

if type(p\_object) == Archer:

self[len(self):] = [p\_object]

else:

raise TypeError(type(p\_object))

except TypeError as Err:

print(f'Invalid type {Err}')

def print\_count(self):

result = 0

for archer in self:

if archer.gender == 'm':

result += 1

print(result)