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. 3341 _____ Перевалов П.И.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2024

Цель работы

Цель этой работы была создание иерархии классов для представления различных персонажей (воинов, магов, лучников) и их списков. Определили основные атрибуты и методы для каждого класса, а также переопределили методы базового класса object для улучшения функциональности и взаимодействия с объектами.

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:

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.

Основные теоретические положения

1. Объекты: В Python все данные и функции обычно объединяются в объекты. Объекты могут быть экземплярами классов или могут быть созданы динамически.

2. Классы: Классы в Python представляют шаблоны для создания объектов. Классы включают в себя атрибуты (данные) и методы (функции).

3. Наследование: Python поддерживает наследование, что позволяет одному классу наследовать атрибуты и методы другого класса. Это позволяет создавать иерархию классов и повторно использовать код.

4. Инкапсуляция: Python поддерживает инкапсуляцию, что означает, что данные класса защищены от прямого доступа извне. Доступ к данным должен осуществляться через методы, определенные в классе.

5. Полиморфизм: Python поддерживает полиморфизм, который позволяет объектам с одним интерфейсом иметь различную реализацию. Это позволяет использовать объекты разных классов с одним и тем же интерфейсом во время выполнения.

6. Методы: Методы в Python могут быть экземплярными (связанными с экземпляром объекта) или статическими (связанными с классом). Статические методы не имеют доступа к экземплярным атрибутам и могут использоваться для управления классом в целом.

Выполнение работы

1. Создаем класс Character, который содержит атрибуты gender, age, height, weight. При инициализации объекта проверяем, что gender является "m" или "w", a age, height и weight больше нуля.

2. Создаем класс Warrior, который наследуется от класса Character и добавляет атрибуты forces, physicaldamage, armor. При инициализации объекта вызываем init родительского класса, затем проверяем forces, physicaldamage и armor на положительные значения.

3. Создаем класс Magician, который также наследуется от класса Character и добавляет атрибуты mana, magicdamage. При инициализации объекта вызываем init родительского класса, затем проверяем mana и magicdamage на положительные значения.

4. Создаем класс Archer, аналогично Warrior и Magician, наследуется от Character и добавляет атрибуты forces, physicaldamage, attackrange. При инициализации объекта вызываем init родительского класса, затем проверяем forces, physicaldamage и attackrange на положительные значения.

5. Создаем класс WarriorList, который наследуется от списка и добавляет методы init, append, printcount. Метод append позволяет добавлять только объекты класса Warrior в список.

6. Создаем класс MagicianList, аналогично WarriorList, добавляет методы init, extend, printdamage. Метод extend позволяет добавлять в список только объекты класса Magician.

7. Создаем класс ArcherList, аналогично WarriorList и MagicianList, добавляет методы init, append, printcount. Метод printcount подсчитывает количество мужских арчеров в списке.

Функция check_requirements принимает список переменных variables и проверяет, что все переменные в этом списке являются целыми числами и больше нуля. Если все условия выполняются, функция возвращает True, в противном случае - False.

Этот код реализует иерархию классов персонажей (воин, маг, лучник) и списков для хранения персонажей каждого класса. Каждый класс персонажа имеет свои уникальные атрибуты и методы.

1. Изображение иерархии классов:

```
   Character
  /    |    \
Warrior Magician Archer

WarriorList <-- list
MagicianList <-- list
ArcherList <-- list
```

2. В переопределении методов класса объекта object или других методов:

- Метод `__init__`: переопределен в каждом классе для инициализации атрибутов.

- Метод `__str__`: переопределен для возвращения строкового представления объекта.

3. Метод `str()`: будет использован, когда объект класса вызывается как аргумент функции `str()`, чтобы получить его строковое представление. Метод `__print_damage__()`: предполагается, что это опечатка, вероятно имелось в виду метод `print_damage()`, который должен быть вызван для объектов класса MagicianList.

4. Переопределенные методы класса list для созданных списков не будут работать, т.к. созданные классы WarriorList, MagicianList и ArcherList являются подклассами списка (list), но они не переопределяют поведение всех методов класса list. Например, методы append, extend, print_count могут быть вызваны нормально, так как они определены в наших классах, но такие методы, как clear или pop, которые не переопределены, будут работать как обычно для списка без дополнительной логики, определенной в наших классах.

Пример:

```python
my_warriors = WarriorList()
my_warriors.append(Warrior("m", 30, 180, 80, 100, 50, 10))
print(my_warriors)  # Выведет информацию о воине
print(len(my_warriors))  # Выведет количество воинов
my_warriors.clear()  # AttributeError: 'WarriorList' object has no attribute 'clear'
```

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)
		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 работы основных методов классов

	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()		

2.	try: #неправильные данные для персонажа
	character = Character(-1, 20, 180, 70)
	except (TypeError, ValueError):
	print('OK')
	
	try:
	character = Character('m', -20, 180, 70)
	except (TypeError, ValueError):
	print('OK')
	
	try:
	character = Character('m', 20, -180, 70)
	except (TypeError, ValueError):
	print('OK')
	
	try:
	character = Character('m', 20, 180, -70)
	except (TypeError, ValueError):
	print('OK')
		OK
		OK
		OK
		OK
		OK
		OK
		OK
		OK
		OK
		OK
		OK
		OK
			Проверка неправильных данных

	try: character = Character(1, 20, 180, 70) except (TypeError, ValueError): print('OK') try: character = Character('m', 0, 180, 70) except (TypeError, ValueError): print('OK') try: character = Character('m', 20, 0, 70) except (TypeError, ValueError): print('OK') try: character = Character('m', 20, 180, 0) except (TypeError, ValueError): print('OK') try: character = Character('a', 20, 180, 70) except (TypeError, ValueError): print('OK')		

	try: character = Character('m', 'a', 180, 70) except (TypeError, ValueError): print('OK') try: character = Character('m', 20, 'a', 70) except (TypeError, ValueError): print('OK') try: character = Character('m', 20, 180, 'a') except (TypeError, ValueError): print('OK')		

Выводы

Изучив иерархию классов, поняли, как использовать наследование для создания классов с общими характеристиками, поддерживая при этом уникальные особенности и методы для каждого класса. Также рассмотрели, как переопределить методы базового класса object для более удобной работы с объектами и их строковым представлением.

Методы str() и __print_damage__() могут быть использованы для вывода информации о персонажах, а также о величине наносимого ими урона.

Наконец, установили, что переопределенные методы класса list для созданных подклассов не будут работать для всех методов списка, поскольку классы WarriorList, MagicianList и ArcherList наследуют методы класса list, но не переопределяют все их поведения.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Character:
    def __init__(self, gender, age, height, weight):
        if (gender in {"m", "w"} and checking_conditions([age, height, weight])):
            self.gender = gender
            self.age = age
            self.height = height
            self.weight = weight
        else:
            raise ValueError("Invalid value")


class Warrior(Character):
    def __init__(self, gender, age, height, weight, forces, physical_damage, armor):
        super().__init__(gender, age, height, weight)
        if checking_conditions([forces, physical_damage, armor]):
            self.forces = forces
            self.physical_damage = physical_damage
            self.armor = armor
        else:
            raise ValueError("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):
        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 checking_conditions([mana, magic_damage]):
            self.mana = mana
            self.magic_damage = magic_damage
        else:
            raise ValueError("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.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 checking_conditions([forces, physical_damage,
attack_range]):
                self.forces = forces
                self.physical_damage = physical_damage
                self.attack_range = attack_range
            else:
                raise ValueError("Invalid value")

        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.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__()
            self.name = name

        def append(self, p_object):
            if isinstance(p_object, Warrior):
                super().append(p_object)
            else:
                raise TypeError("Invalid type", 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 element in iterable:
                if isinstance(element, Magician):
                    super().append(element)

        def print_damage(self):
            total_damage = sum(element.magic_damage for element in self)
            print(total_damage)


    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("Invalid type", type(p_object))

        def print_count(self):
            male_archers_number = sum(element.gender == "m" for element
in self)
            print(male_archers_number)

    def checking_conditions(variables):
        if  all(isinstance(param,  int)  for  param  in  variables)  and
all(param > 0 for param in variables):
            return True
        return False
```