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

Студент гр. 3344 _____ Коршунов П.И.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Введение в парадигмы программирования. Освоение парадигм программирования на языке Python.

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

Класс *Character*(родитель для классов *Warrior, Magician, Archer*) – класс для персонажа с полями *gender, age, height, weight*, которые заданы дескрипторами, и методом для строкового представления

Класс *Warrior*(наследник класса *Character*) – класс для воина с полями родительского класса и *forces, physical_damage, armor,* которые заданы дескрипторами, и методом сравнения.

Класс *Magician*(наследник класса *Character*) – класс для мага с полями родительского класса и *mana, magic_damage,* которые заданы дескрипторами, и методом для получения урона за всю ману.

Класс *Archer*(наследник класса *Character*) – класс для лучника с полями родительского класса и *forces, physical_damage, attack_range,* которые заданы дескрипторами, и методом сравнения.

Класс *PositiveInt* (родитель для класса *NotNegativeInt*) – класс дескриптор для работы с целыми положительными значениями, присутствует проверка на корректность данных.

Класс *NotNegativeInt*(наследник класса *PositiveInt*) – класс дескриптор для работы с целыми неотрицательными значениями, присутствует проверка на корректность данных.

Класс *Gender* – класс дескриптор для работы со строками, отвечающими за гендер, присутствует проверка на корректность данных.

Класс *WarriorList*(наследник класса *list*) – класс-список, который может содержать только объекты класса *Warrior,* с методом для отображения длины списка и переопределенным методом *append*.

Класс *MagicianList*(наследник класса *list*) – класс-список, который может содержать только объекты класса *Magician,* с методом для отображения суммы урона магов и переопределенным методом *extend*.

Класс *ArcherList*(наследник класса *list*) – класс-список, который может содержать только объекты класса *Archer,* с методом для отображения количества

элементов списка, у которых полу *gender* равно m, и переопределенным методом *append*.

Методы, которые были переопределены: *__init__, __str__, __eq__, append, extend.*

Метод *__str__* будет использован тогда, когда понадобится строковое представление объекта.

Метод *print_damage* будет использован тогда, когда будет вызван у класса *MagicianList.*

Методы, переопределенные для класса *list*, будут работать, так как это методы класса родителя с дополнительным условием. При добавлении объекта *Warrior* в объект *WarriorList* объект будет успешно добавлен, так как его класс удовлетворяет условию.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	w1 = Warrior('m', 18, 180, 80, 90, 100, 150) w2 = Warrior('m', 30, 200, 80, 90, 100, 150) print(w1 == w2)	True	-
2.	w1 = Magician('m', 18, 180, 80, 90, 100) w2 = Magician('m', 30, 200, 80, 90, 300) l1 = MagicianList('name') l1.extend([w1, w2]) print(w1.magic_damage) l1.print_damage()	100 400	-
3.	w1 = Archer('m', 18, 180, 80, 90, 100, 200) w2 = Archer('w', 30, 200, 80, 90, 300, 500) l1 = ArcherList('name') l1.append(w1) l1.append(w2) l1.print_count()	1	-

Выводы

Были получены базовые знания о парадигмах программирования и их применения в Python.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: Korshunov_Petr_lb1.py

```
class PositiveInt:
    @classmethod
    def verify_num(cls, num):
        if not (isinstance(num, int) and num > 0):
            raise ValueError("Invalid value")

    def __set_name__(self, owner, name):
        self.name = "_" + name

    def __get__(self, instance, owner):
        return getattr(instance, self.name)

    def __set__(self, instance, value):
        self.verify_num(value)
        setattr(instance, self.name, value)


class NotNegativeInt(PositiveInt):
    @classmethod
    def verify_num(cls, num):
        if not (isinstance(num, int) and num >= 0):
            raise ValueError("Invalid value")


class Gender:
    @classmethod
    def verify_gender(cls, gender):
        if gender not in ('m', 'w'):
            raise ValueError("Invalid value")

    def __set_name__(self, owner, name):
        self.name = "_" + name
```

```
    def __get__(self, instance, owner):
        return getattr(instance, self.name)

    def __set__(self, instance, value):
        self.verify_gender(value)
        setattr(instance, self.name, value)


class Character:
    gender = Gender()
    age = PositiveInt()
    height = PositiveInt()
    weight = PositiveInt()
    fields_to_print = {'_gender': "П о л ", '_age': "в о з р а с т ",
'_height': "р о с т ", '_weight': "в е с ",
                       '_forces': "з а п а с  с и л ", '_physical_damage':
"ф и з и ч е с к и й  у р о н ", '_armor': "б р о н я ",
                       '_attack_range': "д а л ь н о с т ь  а т а к и ",
'_mana': "з а п а с  м а н ы ", '_magic_damage': "м а г и ч е с к и й  у р о н
"}

    def __init__(self, gender, age, height, weight):
        self.gender = gender
        self.age = age
        self.height = height
        self.weight = weight

    def __str__(self):
        return       f"{type(self).__name__}:       "       +       f"{',
'.join([self.fields_to_print[k] + str(v) for k, v in self.__dict__.items()
if k in self.fields_to_print])}."


class Warrior(Character):
    forces = PositiveInt()
    physical_damage = PositiveInt()
    armor = PositiveInt()
```

```
    def __init__(self, gender, age, height, weight, forces,
physical_damage, armor):
        super().__init__(gender, age, height, weight)
        self.forces = forces
        self.physical_damage = physical_damage
        self.armor = armor

    def __eq__(self, other):
        return self.forces == other.forces and self.physical_damage ==
other.physical_damage and self.armor == other.armor


class Magician(Character):
    mana = PositiveInt()
    magic_damage = PositiveInt()

    def __init__(self, gender, age, height, weight, mana, magic_damage):
        super().__init__(gender, age, height, weight)
        self.mana = mana
        self.magic_damage = magic_damage

    def __damage__(self):
        return self.magic_damage * self.mana


class Archer(Character):
    forces = PositiveInt()
    physical_damage = PositiveInt()
    attack_range = PositiveInt()

    def __init__(self, gender, age, height, weight, forces,
physical_damage, attack_range):
        super().__init__(gender, age, height, weight)
        self.forces = forces
        self.physical_damage = physical_damage
        self.attack_range = attack_range

    def __eq__(self, other):
```

```
        return self.forces == other.forces and self.physical_damage ==
other.physical_damage 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(f"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):
        super().extend([item for item in iterable if isinstance(item,
Magician)])

    def print_damage(self):
        print(sum([magician.magic_damage for magician 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 {type(p_object)}")

def print_count(self):
    print(len([archer for archer in self if archer.gender == 'm']))
```