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. 3341 _____ Самокрутов А.Р.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2023

Цель работы

Целью данной лабораторной работы является ознакомление с объектно-ориентированной парадигмой программирования в языке, основными её принципами и их воплощением в языкеPython. Для этого необходимо выполнить следующие задачи:

1. Изучить основные принципы объектно-ориентированного программирования — абстракцию, полиморфизм, инкапсуляцию и наследование.
2. Освоить основы работы с классами и атрибутами классов в языке программирования Python
3. Реализовать иерархию классов для представления различных типов объектов в соответствии с заданием.

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:

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.

Основные теоретические положения

Объект — конкретная сущность некоторой предметной области., обозначает некоторую абстракцию (обобщение).

Класс — это тип объекта. Класс описывает общее поведение: общие черты, свойства и характеристики, а также общие действия, функции, которые можно выполнять над объектами класса.

Поля классов — это общие свойства, характеристики классов. Методы классов — это функции для работы с объектами классов. Методы, как и поля, определены в самих классах. И поля, и методы классов иногда называют одним словом: атрибуты класса. Вызов методов объекта класса и обращение к полям объекта класса выполняются с использованием символа “.”.

Практически всё, что может встретиться в языке Python, является объектом: числа, строки и даже функции. Все они имеют свой собственный класс, в котором определены атрибуты: поля и методы.

Объектно-ориентированная парадигма базируется на нескольких принципах: наследование, инкапсуляция, полиморфизм.

Наследование — специальный механизм, при котором можно расширять классы, усложняя их функциональность.

Под инкапсуляцией часто понимают сокрытие внутренней реализации от пользователя. В других языках программирования это достигается использованием модификаторов доступа; таким образом, в описании класса можно указать, какой атрибут будет доступен извне, а какой — нет. В языке Python этот механизм лишь указывает, что атрибут не должен быть изменен.

В некоторых языках существует возможность создать несколько функций с одинаковым именем, но разными типами аргументов. Это называется перегрузкой функций. В языке Python невозможно воспользоваться таким механизмом, поскольку в языке нет объявления типа и нельзя создать функцию с тем же именем в той же области видимости (сохранится только последнее определение функции). В языке Python полиморфность функций выражается в том, что они могут работать с разными типами данных.

Выполнение работы

Создаётся базовый класс Character, который содержит поля gender, age, height, weight. В конструкторе производится проверка того, что поле gender является одной из строк *‘m’* и *‘w’*, а остальные поля — положительные целые числа, если это не так, то метод выбрасывает исключение *ValueError*.

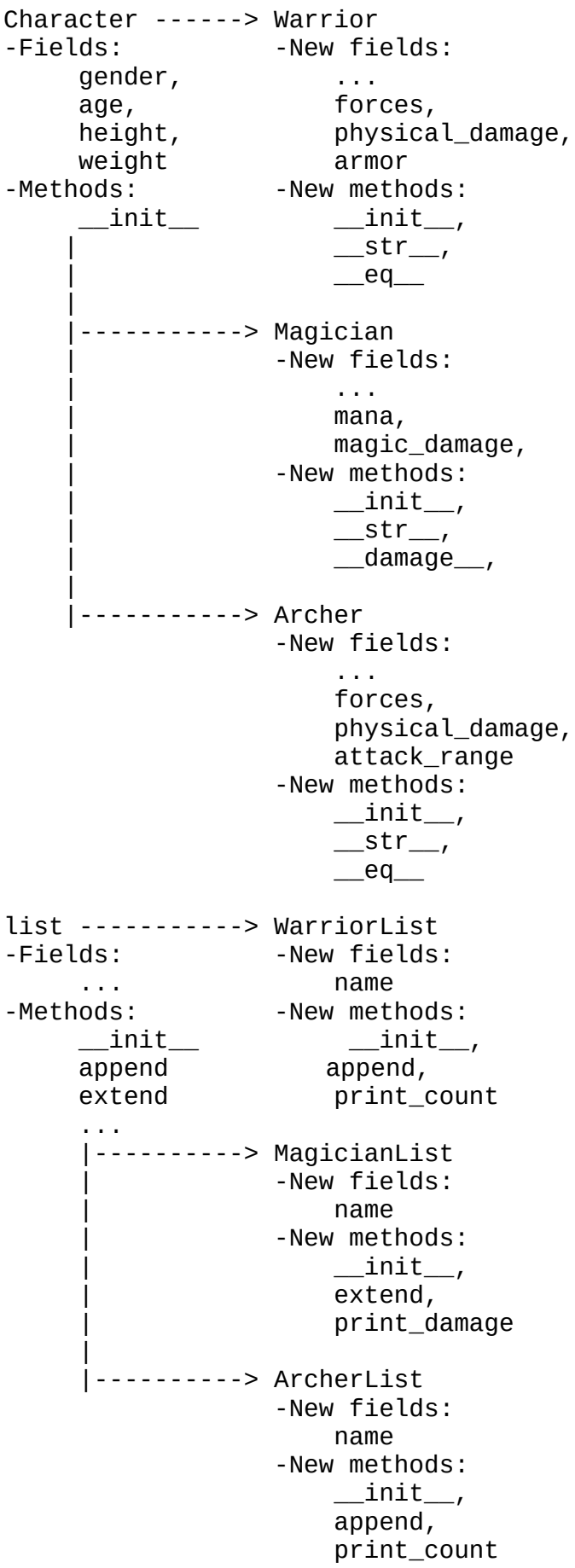
Создаётся класс *Warrior* — наследник класса *Character*. В нём появляются дополнительные поля *forces*, *physical_damage*, *armor*. При инициализации объекта вызывается конструктор класса-родителя, затем инициализируются новые поля. Если они не являются целыми положительными числами, метод выбрасывает ошибку *ValueError*. Аналогично создаются классы-наследники от *Character*: *Magician* (новые поля *mana*, *magic_damage*) и *Archer* (новые поля *forces*, *physical_damage*, *attack_range*).

Создаётся класс *WarriorList* — наследник класса *list*, в котором переопределяются методы *__init__* и *append* и добавляется метод *print_count*. Метод *__init__* вызывает конструктор класса-родителя и присваивает новому полю name значение аргумента *name*. Метод *append* проверяет добавляемый объект на принадлежность к классу Warrior и добавляет объект либо вызывает ошибку *ValueError* в зависимости от результата проверки. Метод *print*_count выводит на экран результат метода *__len__*. Аналогично описан класс *ArcherList*.

Создаётся класс *MagicianList* — наследник класса *list*, котором переопределяются методы *__init__* и *extend* и добавляется метод *print_damage*. Конструктор класса аналогичен таковым в классах, описанных выше. Метод *extend* проверяет каждый из элементов итерируемого объекта на принадлежность к классу *Magician* и либо добавляет элемент в список, либо игнорирует его. Метод *print_damage* выводит на экран сумму полей *magic_damage* всех элементов списка.

Код программы реализует иерархию классов персонажей (*Character*, *Warrior*, *Magician*, *Archer*) и списков каждого из типов персонажей (*WarriorList*, *MagicianList*, *ArcherList*). Каждый класс имеет свои уникальные поля и методы.

Схематичное изображение иерархии классов:



Переопределённые методы:

- *__init__*: переопределён в каждом классе для инициализации полей.
- *__str__*: переопределён в классах Warrior, Magician, Archer для возвращения строкового представления объекта.
- *__eq__*: переопределён в классах Warrior, Archer для сравнения двух объектов на равенство.
- *append*: переопределён в классах *WarriorList*, *ArcherList* для добавления в список объекта класса *Warrior* или *Archer* соответсвенно.
- *extend*: переопределён в классе *MagicianList* для добавления в список каждого из объектов класса *Magician* в итерируемом объекте.

Метод *__str__()* будет использован всякий раз, когда объект класса вызывается как аргумент функции *str()* для получения его строкового представления. Метод *print_damage()* будет вызван для вывода на экран суммарного урона всех магов (элементы списка *MagicianList*).

Переопределённые методы класса *list* в классах *WarriorList*, *MagicianList*, *ArcherList* будут работать должным образом только для объектов соответствующего класса. Методы класса *list*, не переопределённые в дочерних классах, будут работать как для обычного списка без дополнительной логики, определённой в списках-наследниках. Так, метод *pop()* будет работать корректно, а методы *append* в списке *MagicianList* и *extend* в списках *WarriorList* и *ArcherList* будут работать неправильно: они смогут добавить в список объекты, принадлежащие к других классам, что не соответствует логике созданных списков.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы

Были изучены классы в языке программирования Python, изучены основы наследования классов — одного из базовых принципов объектно-ориентированного программирования. Были освоены основные понятия, которыми оперирует ООП. На языке программирования Python была написана программа, реализующая иерархию классов, наследование, переопределение методов базовых классов.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Character:
    def __init__(self, gender, age, height, weight):
        if (gender in ['m', 'w'] and all(isinstance(parameter, int) for parameter in [age, height, weight])
                and all(parameter > 0 for parameter in [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):
        if (all(isinstance(parameter, int) for parameter in [forces, physical_damage, armor])
                and all(parameter > 0 for parameter in [forces, physical_damage, armor])):
            super().__init__(gender, age, height, weight)
            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):
        if isinstance(other, Warrior):
            return self.physical_damage == other.physical_damage \
                and self.forces == other.forces \
                and self.armor == other.armor
        else:
            return False


class Magician(Character):
    def __init__(self, gender, age, height, weight, mana, magic_damage):
        if (all(isinstance(parameter, int) for parameter in [mana, magic_damage])
                and all(parameter > 0 for parameter in [mana, magic_damage])):
            super().__init__(gender, age, height, weight)
            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):
                if (all(isinstance(parameter, int) for parameter in [forces, physical_damage, attack_range])
                        and all(parameter > 0 for parameter in [forces, physical_damage, attack_range])):
                    super().__init__(gender, age, height, weight)
                    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):
            if isinstance(other, Archer):
                return self.physical_damage == other.physical_damage \
                    and self.forces == other.forces \
                    and self.attack_range == other.attack_range
            else:
                return False


    class WarriorList(list):
        def __init__(self, name):
            super().__init__(self)
            self.name = str(name)

        def append(self, war):
            if isinstance(war, Warrior):
                self += [war]
            else:
                raise TypeError(f'Invalid type {type(war)}')


        def print_count(self):
            print(self.__len__())


    class MagicianList(list):
```

```
    def __init__(self, name):
        super().__init__(self)
        self.name = str(name)

    def extend(self, mags):
        temp = []
        for mag in mags:
            if isinstance(mag, Magician):
                temp.append(mag)
        super().extend(temp)

    def print_damage(self):
        print(sum([mag.magic_damage for mag in self]))


class ArcherList(list):
    def __init__(self, name):
        super().__init__(self)
        self.name = str(name)

    def append(self, arch):
        if isinstance(arch, Archer):
            self += [arch]
        else:
            raise TypeError(f'Invalid type {type(arch)}')

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

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	character = Character('m', 20, 180, 70) print(character.gender, character.age, character.height, character.weight)	m 20 180 70	Проверка корректной работы базового класса.
2.	character = Character('m', -1, 1, 1) print(character.gender, character.age, character.height, character.weight)	ValueError: Invalid value	Проверка обработки неправильных входных данных в базовом классе.
3.	warrior1 = Warrior('m', 20, 180, 70, 50, 100, 30) warrior2 = Warrior('w', 20, 180, 70, 50, 100, 30) print(warrior1.__str__()) print(str(warrior2)) print(warrior1.__eq__(warrior2)) print(warrior1 == warrior2) magician1 = Magician('m', 20, 180, 70, 60, 110) magician2 = Magician('m', 40, 180, 70, 60, 110) print(magician1.__str__()) print(str(magician2)) print(magician1.__damage__())	Warrior: Пол m, возраст 20, рост 180, вес 70, запас сил 50, физический урон 100, броня 30. Warrior: Пол w, возраст 20, рост 180, вес 70, запас сил 50, физический урон 100, броня 30. True True Magician: Пол m, возраст 20, рост 180, вес 70, запас маны 60, магический урон 110. Magician: Пол m, возраст 40, рост 180, вес 70, запас маны 60, магический урон	Проверка корректной работы классов, наследованных от базового класса персонажа и их методов.

	archer1 = Archer('m', 20, 180, 70, 60, 95, 50) archer2 = Archer('m', 20, 3000, 70, 60, 95, 50) print(archer1.__str__()) print(str(archer2)) print(archer1.__eq__(archer2)) print(archer1 == archer2)	110. 6600 Archer: Пол m, возраст 20, рост 180, вес 70, запас сил 60, физический урон 95, дальность атаки 50. Archer: Пол m, возраст 20, рост 3000, вес 70, запас сил 60, физический урон 95, дальность атаки 50. True True	

4	warrior1 = Warrior('m', 20, 180, 70, 50, 100, 30)
	warrior2 = Warrior('w', 20, 180, 70, 50, 100, 30)
	warrior_list = WarriorList(Warrior)
	warrior_list.append(warrior1)
	warrior_list.append(warrior2)
	warrior_list.print_count()
	try:
	  warrior_list.append(1)
	except:
	  print('Error! It should be an error!')
	
	magician1 = Magician('m', 20, 180, 70, 60, 110)
	magician2 = Magician('m', 40, 180, 70, 60, 110)
	mag_list =
		2
		Error! It should be an error!
		220
		2
		Error! It should be an error!
			Проверка корректной работы классов, наследованных от list.

	MagicianList(Magician) mag_list.extend([magician1, magician2]) mag_list.print_damage() archer1 = Archer('m', 20, 180, 70, 60, 95, 50) archer2 = Archer('m', 20, 3000, 70, 60, 95, 50) archer_list = ArcherList(Archer) archer_list.append(archer1) archer_list.append(archer2) archer_list.print_count() try: archer_list.append(1) except: print('Error! It should be an error!')		