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

Студентка гр. 3341 Чинаева М.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

Объектно-ориентированное программирование — методология или стиль программирования на основе описания типов/моделей предметной области и их взаимодействия, представленных порождением из прототипов или как экземпляры классов, которые образуют иерархию наследования.

Идеологически, ООП — подход к программированию как к моделированию информационных объектов, решающий на более высоком абстрактном уровне основную задачу структурного программирования — структурирование информации с точки зрения управляемости. Это позволяет управлять самим процессом моделирования и реализовывать крупные программные проекты.

Класс — это тип данных, созданный пользователем. Он содержит разные свойства и методы, как, например, тип String или Int.

Объект — это экземпляр класса, или его копия, которая находится в памяти компьютера.

Выполнение работы

Создаётся базовый класс Character, содержащий поля gender, age, height, weight. В конструкторе производится проверка полей на удовлетворение требований (поле gender – одна из строк *'m'* и *'w'*, а остальные поля — положительные целые числа). Метод выбрасывает исключение *ValueError*, если проверка неудачная.

Создаются классы – наследники класса *Character:*

1. *Warrior*

Конструктор аналогичен классу-родителю, но с дополнительными полями *forces*, *physical_damage*, *armor* и их проверкой.

2. *Magician*

Конструктор аналогичен классу-родителю, но с дополнительными полями *mana*, *magic_damage* и их проверкой.

3. Archer

Конструктор аналогичен классу-родителю, но с дополнительными полями *physical_damage*, *attack_range* и их проверкой.

Создаётся классы-наследники класса *list*:

1. *WarriorList*

Метод *__init__* вызывает конструктор класса-родителя и присваивает полю name значение аргумента *name*.

Метод *append* проверяет добавляемый в список объект на принадлежность к классу Warrior и либо добавляет объект, если он принадлежит данному классу, либо вызывает ошибку *ValueError*.

Метод *print*_count выводит на экран результат метода *__len__*.

2. *ArcherList*.

Методы *__init__* и *append* аналогичны классу *WarriorList*

Метод print_count считает и выводит количество элементов списка, у которых строка gender равна 'm'

3.*MagicianList*

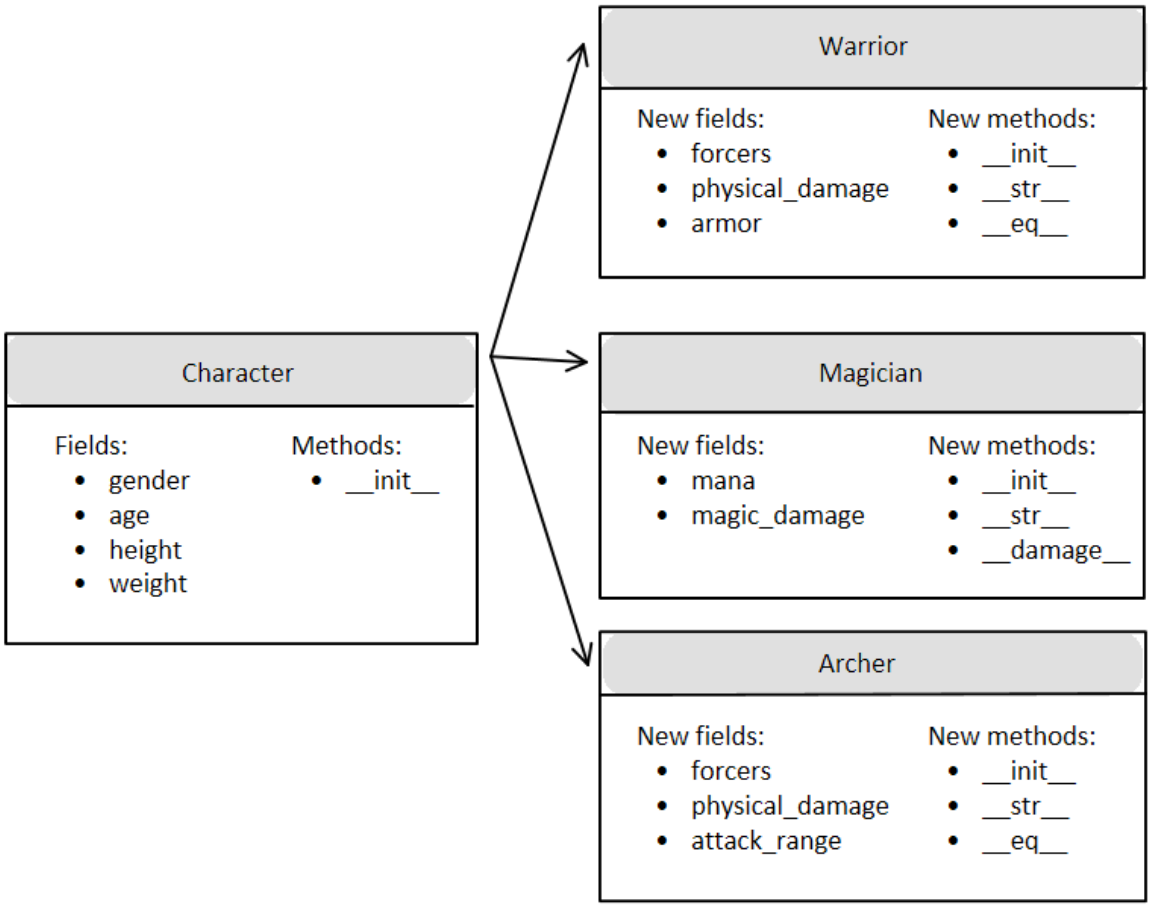
Метод *__init__* аналогичен классу *WarriorList*

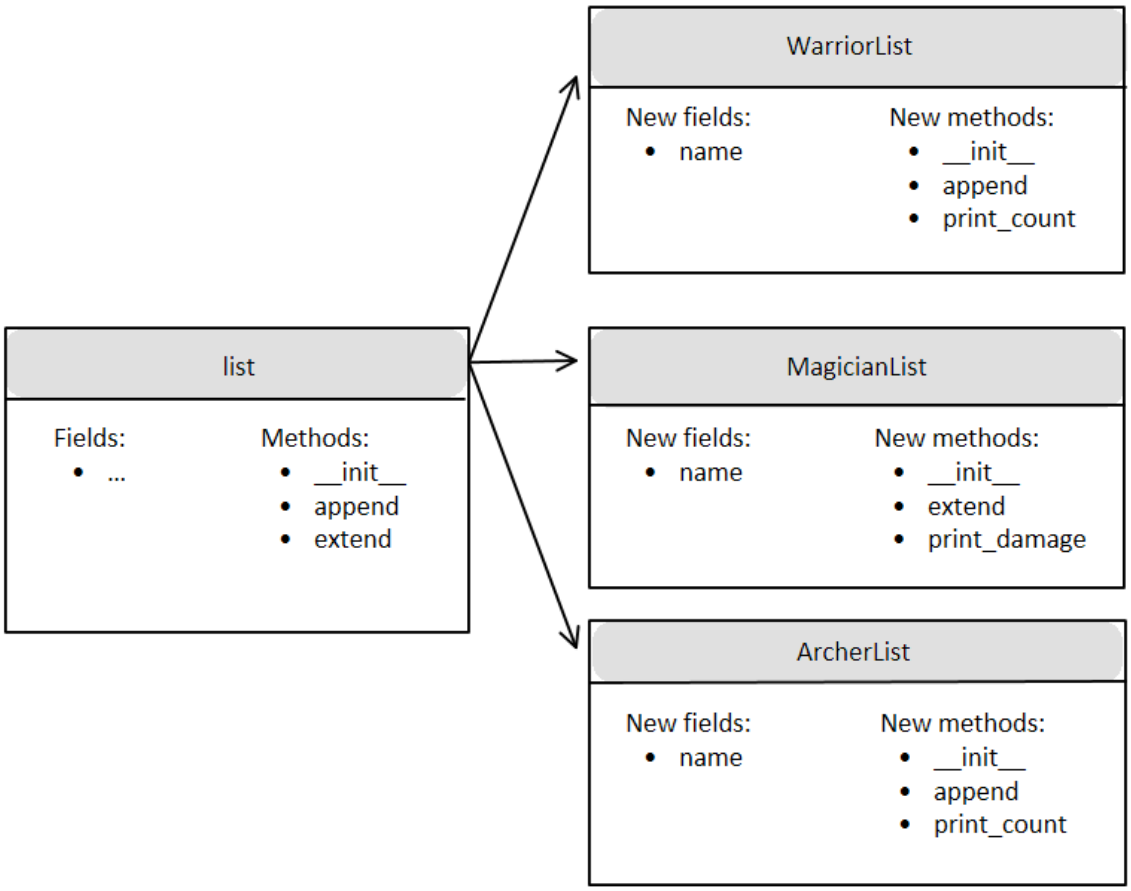
Метод *extend* проверяет каждый из элементов итерируемого объекта на принадлежность к классу *Magician* и все подходящие записывает в исходный список.

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:

1. *__init__*: переопределён в каждом классе для инициализации полей.
2. *__str__*: переопределён в классах Warrior, Magician, Archer для возвращения описания объекта класса в виде строки.
3. *__eq__*: переопределён в классах Warrior, Archer для сравнения двух объектов классов на равенство.
4. *append*: переопределён в классах *WarriorList*, *ArcherList* для добавления в список объекта класса *Warrior* или *Archer* соответственно.
5. *extend*: переопределён в классе *MagicianList* для добавления в список только объектов класса Magician из итерируемого объекта.

Метод *__str__()* будет использован, когда объект класса вызывается как аргумент функции *str()*. Метод *print_damage()* будет вызван для вывода на экран суммарного урона всех магов.

Переопределённые методы класса *list* в классах *WarriorList*, *MagicianList*, *ArcherList* будут работать только для объектов соответствующего класса. Методы класса *list*, не переопределённые в классах-наследниках, будут работать как для обычного списка.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы

Была изучена объектно-ориентированная парадигма программирования. 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(x, int) and
x>0) for x 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 (gender in ['m', 'w'] and all ((isinstance(x, int) and
x>0) for x in [age, height, weight, forces, physical_damage, armor])):
            self.gender = gender
            self.age = age
            self.height = height
            self.weight = 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):
        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):
        if (gender in ['m', 'w'] and all ((isinstance(x, int) and
x>0) for x in [age, height, weight, mana, magic_damage])):
            self.gender = gender
            self.age = age
            self.height = height
            self.weight = 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}, рос
т {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 (gender in ['m', 'w'] and all ((isinstance(x, int) and
x>0) for x in [age, height, weight, forces, physical_damage,
attack_range])):
                self.gender = gender
                self.age = age
                self.height = height
                self.weight = 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):
            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(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):
            magican_list = [element for element in iterable if
isinstance(element, Magician)]
            super().extend(magican_list)
```

```
    def print_damage(self):
        all_damage = 0
        for element in self:
            all_damage+=element.magic_damage
        print(all_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(f"Invalid type {type(p_object)}")

    def print_count(self):
        count_men = 0
        for element in self:
            if element.gender == 'm':
                count_men+=1
        print(count_men)
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	character = Character('m', 18, 186, 80) warrior = Warrior('m', 23, 170, 65, 100, 100, 20) magician = Magician('w', 21, 170, 60, 10, 41) archer = Archer('w', 20, 160, 50, 30, 100, 150) print(character.gender, character.age, character.height, character.weight) print(warrior.__str__()) print(magician.__str__()) print(archer.__str__())	m 18 186 80 Warrior: Пол m, возраст 23, рост 170, вес 65, запас сил 100, физический урон 100, броня 20. Magician: Пол w, возраст 21, рост 170, вес 60, запас маны 10, магический урон 41. Archer: Пол w, возраст 20, рост 160, вес 50, запас сил 30, физический урон 100, дальность атаки 150.	Проверка корректной работы классов.
2.	character = Character('c', 18, 186, 80)print(character.gender, character.age, character.height, character.weight)	ValueError: Invalid value	Проверка обработки неправильных входных данных в базовом классе.
3.	warrior = Warrior('m', -23, 170, 65, 100, 100, 20) print(warrior.__str__())	ValueError: Invalid value	Проверка обработки неправильных входных данных в классе Warrior.
4.	magician = Magician('man', -21, -170, -60, 0, 0) print(magician.__str__())	ValueError: Invalid value	Проверка обработки неправильных входных данных в классе Magician.

5.	warrior1 = Warrior('m', 23, 170, 65, 100, 100, 20) warrior2 = Warrior('m', 23, 170, 65, 100, 100, 20) warrior3 = Warrior('w', 23, 172, 66, 10, 10, 30) magician = Magician('w', 21, 170, 60, 10, 41) archer1 = Archer('w', 20, 160, 50, 30, 100, 150) archer2 = Archer('w', 20, 160, 50, 30, 100, 150) archer3 = Archer('m', 2, 16, 5, 3, 10, 15) print(str(warrior2)) print(warrior1.__eq__(warrior2)) print(warrior1 == warrior2) print(warrior1.__eq__(warrior3)) print(warrior1 == warrior3) print(magician.__damage__()) print(str(archer2)) print(archer1.__eq__(archer2)) print(archer1 == archer2) print(archer1.__eq__(archer3)) print(archer1 == archer3)	Warrior: Пол m, возраст 23, рост 170, вес 65, запас сил 100, физический урон 100, броня 20. True True False False 410 Archer: Пол w, возраст 20, рост 160, вес 50, запас сил 30, физический урон 100, дальность атаки 150. True True False False	Проверка корректной работы классов и их методов.

6.	character = Character('m', 18, 186, 80) warrior1 = Warrior('m', 23, 170, 65, 100, 100, 20) warrior2 = Warrior('m', 23, 170, 65, 100, 100, 20) archer1 = Archer('w', 20, 160, 50, 30, 100, 150) warrior_list = WarriorList(Warrior) warrior_list.append(warrior 1) warrior_list.append(warrior 2) warrior_list.print_count() warrior_list.append(archer1)	2 Invalid type <class '__main__.Archer'>	Проверка корректной работы классов, наследованных от list.
7.	magician1 = Magician('w', 21, 170, 60, 10, 41) magician2 = Magician('w', 21, 170, 60, 30, 51) mag_list = MagicianList(Magician) mag_list.extend([magician1, magician2]) mag_list.print_damage()	92	Проверка корректной работы классов, наследованных от list.
8.	warrior3 = Warrior('w', 23, 172, 66, 10, 10, 30) archer1 = Archer('w', 20, 160, 50, 30, 100, 150) archer2 = Archer('w', 20, 160, 50, 30, 100, 150) archer3 = Archer('m', 2, 16, 5, 3, 10, 15)	1 Invalid type <class '__main__.Warrior'>	Проверка корректной работы классов, наследованных от list.

	archer_list = ArcherList(Archer) archer_list.append(archer1) archer_list.append(archer3) archer_list.print_count() archer_list.append(warrior3)		