

**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. 3343

Какира У.Н.

Преподаватель

Иванов Д.В.

Санкт-Петербург

2024

## **Цель работы**

Рассмотреть парадигмы программирования на примере принципов наследования классов и переопределения методов из 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:

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.

## Выполнение работы

В классе `Warrior` были переопределены методы `__str__`, отвечающий за строковое представление имени объекта и всех его характеристик, и `__eq__`, отвечающее за сравнение с другими объектами этого же класса, вызван метод `super().__init__()`.

В классе `Magician` были переопределены методы `__str__`, отвечающий за строковое представление имени объекта и всех его характеристик, и `__damage__`, подсчитывающий общий урон всех магов, вызван метод `super().__init__()`.

В классе `Archer` были переопределены методы `__str__`, отвечающий за строковое представление имени объекта и всех его характеристик, и `__eq__`, отвечающее за сравнение с другими объектами этого же класса, вызван метод `super().__init__()`.

В классе `WarriorList` был переопределен метод `append(p_object)`, добавляющий элемент в конец списка, если он соответствует этому классу, иначе выводит соответствующую ошибку и определен метод `print_count`, который выводит количество имеющихся воинов.

В классе `MagicianList` был переопределен метод `__extend__(iterable)`, добавляющий элемент в конец списка, если он соответствует этому классу, иначе выводит соответствующую ошибку и определен метод `print_damage`, который выводит общий магический урон.

В классе `ArcherList` был переопределен метод `append(p_object)`, добавляющий элемент в конец списка, если он соответствует этому классу, иначе выводит соответствующую ошибку и определен метод `print_count`, который выводит количество лучников мужского пола.



## Тестирование

Здесь результаты тестирования, которые помещаются на одну страницу.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	<pre>character = Character('m', 20, 180, 70) #персонаж  print(character.ge nder, 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.ge nder, warrior1.age, warrior1.height, warrior1.weight, warrior1.forces, warrior1.physical_dama ge, warrior1.armor)</pre>	<pre>m 20 180 70 m 20 180 70 50 100 30  Warrior:   Pm, возраст 20, рост 180, вес 70, запас сил 50, физический урон 100, броня 30.  True m 20 180 70 60 110  Magician: P  m, возраст 20, рост 180, вес 70, запас маны 60, магический урон 110. 6600 m 20 180 70 60 95 50  Archer:   W  m, возраст 20, рост 180, вес 70, запас сил 60,</pre>	Ответ верный

<pre> 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.gende r, mag1.age, mag1.height, mag1.weight, mag1.mana, mag1.magic_damage) print(mag1.__str__ __()) print(mag1.__da mage__())  archer1 = Archer('m', 20, 180, 70, 60, 95, 50) #лучник archer2 = Archer('m', 20, 180, 70, 60, 95, 50) </pre>	<pre> физический урон 95, дальность атаки 50.  True 2 220  2 </pre>	
---	---	--

<pre>        print(archer1.gen der, archer1.age, archer1.height, archer1.weight, archer1.forces, archer1.physical_damag e, archer1.attack_range)         print(archer1.__st r__())         print(archer1.__e q__(archer2))          warrior_list = WarriorList(Warrior) #СПИСОК ВОИНОВ         warrior_list.appe nd(warrior1)         warrior_list.appe nd(warrior2)         warrior_list.print _count()          mag_list = MagicianList(Magician ) #СПИСОК МАГОВ         mag_list.extend([ mag1, mag2])</pre>		
---	--	--

<pre>mag_list.print_da mage()  archer_list = ArcherList(Archer) #список лучников archer_list.appen d(archer1) archer_list.appen d(archer2) archer_list.print_count( )</pre>		
---	--	--

## **Выводы.**

Были изучены принципы объектно-ориентированного программирования. Закреплены навыки создания классов, их наследников, создания и переопределения методов. Была построена иерархия классов и созданы методы для работы с ними.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
class Character:
    def __init__(self, gender, age, height, weight):
        if (gender not in ('m', 'w')) or (not isinstance(age, int))
or (not isinstance(height, int)) or (not isinstance(weight, int)) or
(age <= 0) or (height <= 0) or (weight <= 0):
            raise ValueError("Invalid value")
        self.gender = gender
        self.age = age
        self.height = height
        self.weight = weight

class Warrior(Character):
    def __init__(self, gender, age, height, weight, forces,
physical_damage, armor):
        super().__init__(gender, age, height, weight)
        if (not isinstance(forces, int)) or (not
isinstance(physical_damage, int)) or (not isinstance(armor, int)) or
(forces <= 0) or (physical_damage <= 0) or (armor <= 0):
            raise ValueError('Invalid value')
        self.forces = forces
        self.physical_damage = physical_damage
        self.armor = armor

    def __str__(self):
        return f'Warrior: Пол {self.gender}, Возраст
{self.age}, рост {self.height}, Вес {self.weight}, запасы си
л {self.forces}, физический урон {self.physical_damage},
броня {self.armor}.'

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

class Magician(Character):
    def __init__(self, gender, age, height, weight, mana,
magic_damage):
        super().__init__(gender, age, height, weight)
        if (not isinstance(mana, int)) or (not
isinstance(magic_damage, int)) or (mana <= 0) or (magic_damage <=
0):
            raise ValueError("Invalid value")
        self.mana = mana
        self.magic_damage = magic_damage

    def __str__(self):
        return f'Magician: Пол {self.gender}, Возраст
{self.age}, рост {self.height}, Вес {self.weight}, запасы ма
ны {self.mana}, магический урон {self.magic_damage}.'
```

```

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

class Archer(Character):
    def __init__(self, gender, age, height, weight, forces,
physical_damage, attack_range):
        super().__init__(gender, age, height, weight)
        if (not isinstance(forces, int)) or (not
isinstance(physical_damage, int)) or (not isinstance(attack_range,
int)) or (forces <= 0) or (physical_damage <= 0) or (attack_range <=
0):
            raise ValueError("Invalid value")
        self.forces = forces
        self.physical_damage = physical_damage
        self.attack_range = attack_range

    def __str__(self):
        return f'Archer: Пол {self.gender}, Возраст
{self.age}, рост {self.height}, Вес {self.weight}, запаса си
л {self.forces}, физический урон {self.physical_damage},
дальность атаки {self.attack_range}.'

    def __eq__(self, other):
        return (int(self.forces) == int(other.forces)) and
(int(self.physical_damage) == int(other.physical_damage)) and
(int(self.attack_range) == int(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
{}".format(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 i in range(len(iterable)):
            if isinstance(iterable[i], Magician):
                super().append(iterable[i])

    def print_damage(self):
        team_damage = 0

```

```

        for i in self:
            team_damage += int(i.magic_damage)
        print(team_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
{}".format(type(p_object)))

    def print_count(self):
        num = 0
        for x in self:
            if x.gender == 'm':
                num +=1
        print(num)

```