**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.

Вариант 2

|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Поддубный В. А |
| Преподаватель |  | Иванов Д. В. |

Санкт-Петербург

2024

## Цель работы

* Освоить принципы объектно-ориентированного программирования (ООП) в Python:
  + Работа с классами, создание методов и функций.
  + Наследование и переопределение методов.
  + Использование super().
* Создать программу, моделирующую работу с геометрическими фигурами.

## Задание

**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**

**Переопределенные методы:**

* + \_\_init\_\_(): В каждом классе переопределен для инициализации атрибутов объекта.
  + \_\_str\_\_(): Переопределен в классах Warrior, Magician и Archer для предоставления строкового представления объектов.
  + \_\_eq\_\_(): Переопределен в классах Warrior и Archer для сравнения объектов по определенным атрибутам.
  + append(): Переопределен в классах WarriorList и ArcherList для добавления объектов только определенного типа.
  + extend(): Переопределен в классе MagicianList для добавления объектов только определенного типа.

**Использование методов \_\_str\_\_() и \_\_print\_damage\_\_():**

* **\_\_str\_\_():** Вызывается неявно при преобразовании объекта класса в строку, например, при использовании функции print() или при конкатенации со строкой.
* **\_\_print\_damage\_\_():** Метод с таким именем в коде отсутствует. Вероятно, имелся в виду метод print\_damage() класса MagicianList. Он вызывается явно для вывода суммарного магического урона всех магов в списке.

**Работа переопределенных методов класса list:**

Да, переопределенные методы класса list будут работать для созданных списков WarriorList, MagicianList и ArcherList, так как эти классы наследуют функциональность от list.

Переопределение методов append и extend позволяет контролировать типы объектов, добавляемых в списки, что обеспечивает безопасность типов.

Отработает корректно:

warriors.append(Warrior('m', 25, 180, 80, 100, 20, 50))

Вызовет TypeError:

warriors.append(Magician('w', 20, 165, 55, 150, 30))

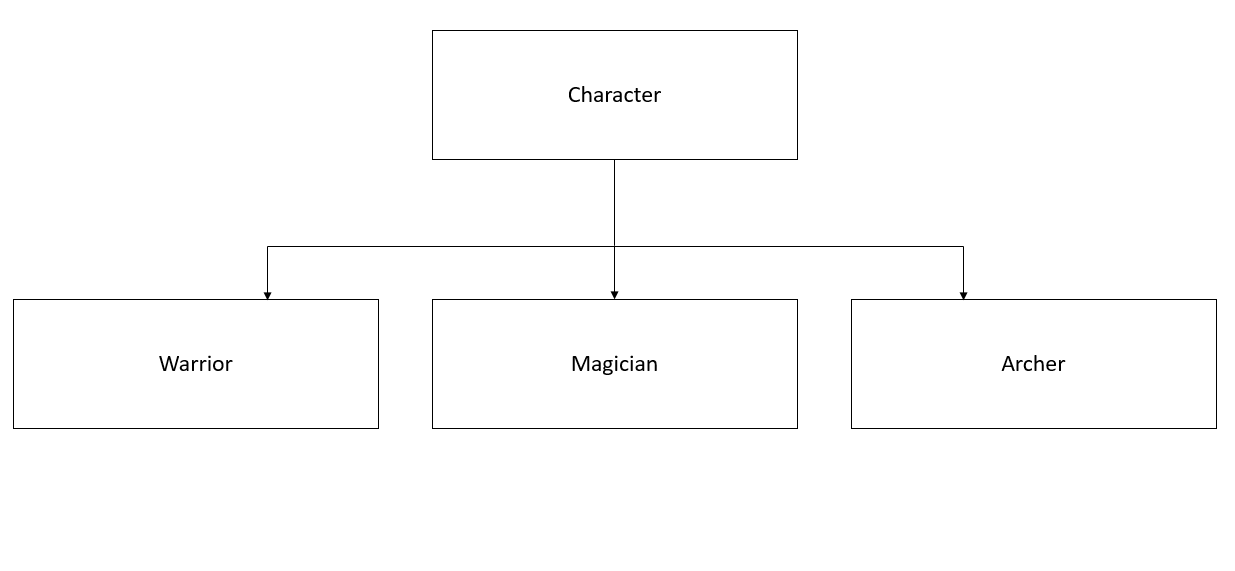


Рисунок 1 – Иерархия классов персонажей

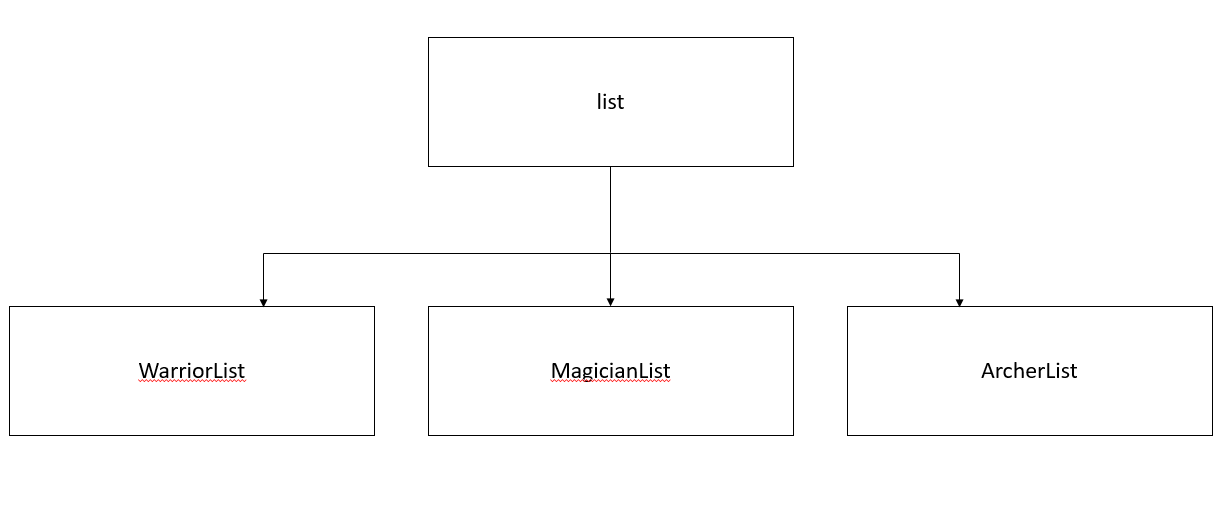


Рисунок 2 – Иерархия классов списков персонажей

## Выводы

В рамках исследования были детально изучены следующие аспекты объектно-ориентированного программирования:

* **Механизм наследования**, позволяющий создавать новые классы на основе существующих, наследуя их атрибуты и методы. При этом допускается наследование как от одного, так и от нескольких родительских классов.
* **Переопределение методов**, дающее возможность изменять поведение унаследованных методов в дочерних классах, адаптируя их к специфике новых объектов.
* **Использование функции super()**, предоставляющей доступ к методам родительского класса из методов дочернего класса. Это особенно полезно при переопределении методов, когда требуется расширить функциональность родительского метода, сохраняя его базовую реализацию.

# Приложение А Исходный код программы

Название файла: main.py

class Character:  
  
 def \_\_init\_\_(self, gender, age, height, weight):  
 if gender not in ['m', 'w']:  
 raise ValueError('Invalid value')  
 if not isinstance(age, int) or age <= 0:  
 raise ValueError('Invalid value')  
 if not isinstance(height, int) or height <= 0:  
 raise ValueError('Invalid value')  
 if not isinstance(weight, int) 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 forces <= 0:  
 raise ValueError('Invalid value')  
 if not isinstance(physical\_damage, int) or physical\_damage <= 0:  
 raise ValueError('Invalid value')  
 if not isinstance(armor, int) 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}, 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 not isinstance(mana, int) or mana <= 0:  
 raise ValueError('Invalid value')  
 if not isinstance(magic\_damage, int) 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}, 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):  
 super().\_\_init\_\_(gender, age, height, weight)  
  
 if not isinstance(forces, int) or forces <= 0:  
 raise ValueError('Invalid value')  
 if not isinstance(physical\_damage, int) or physical\_damage <= 0:  
 raise ValueError('Invalid value')  
 if not isinstance(attack\_range, int) 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}, 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).\_\_name\_\_}')  
  
 def print\_count(self):  
 print(len(self))  
  
  
class MagicianList(list):  
  
 def \_\_init\_\_(self, name):  
 super().\_\_init\_\_()  
 self.name = name  
  
 def extend(self, iterable):  
 for el in iterable:  
 if isinstance(el, Magician):  
 self.append(el)  
  
 def print\_damage(self):  
 damage = 0  
 for magician in self:  
 damage += magician.magic\_damage  
 print(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).\_\_name\_\_}')  
  
 def print\_count(self):  
 count = 0  
 for archer in self:  
 if archer.gender == 'm':  
 count+=1  
 print(count)