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 _____ Вердин К.К.

Преподаватель _____ Иванов.Д.В

Санкт-Петербург
2024

Цель работы

Получить представление о работе ООП в языке Python.

Задание.

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.

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

1)Ниже представлено изображение иерархии классов (см рис.1)

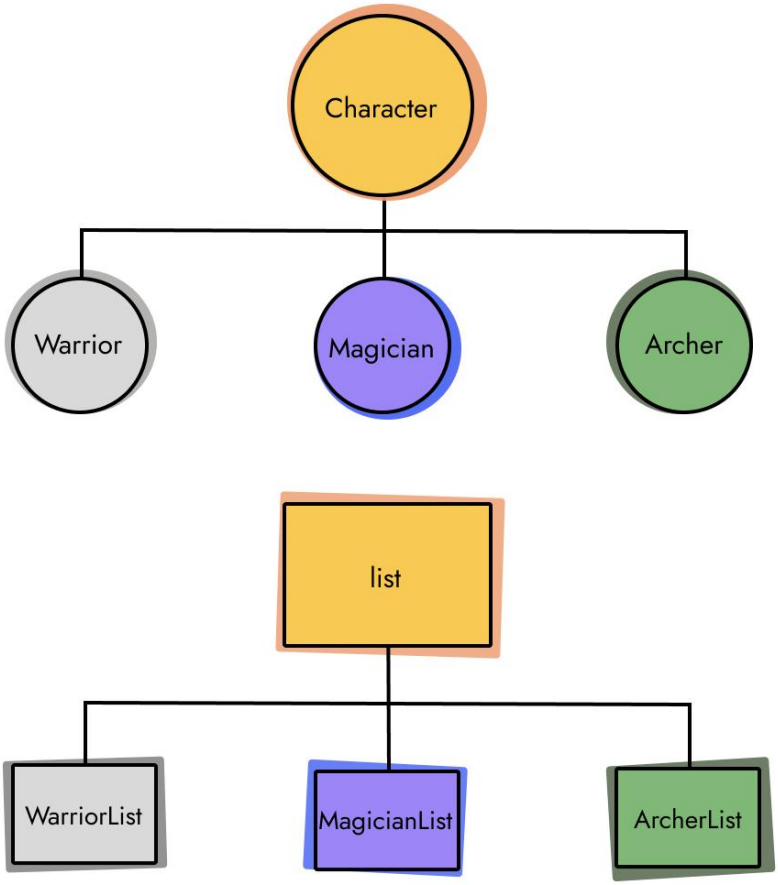

Рисунок 1 – Изображение иерархии классов

2) Методы класса object, которые переопределены:

__init__

__str__

__eq__

Дополнительные методы, которые переопределены:

append в классе WarriorList и ArcherList

extend в классе MagicianList

Метод __str__() будет использоваться при представлении объекта в виде строки, например при применения к объекту методов str() и print().

Переопределенные методы класса list для созданных списков будут работать, так как это такие же методы, только с дополнительной проверкой.

Например, метод append класса ArcherList делает то же самое, что и стандартный метод append, просто проверяет перед этим, принадлежит ли объект классу Archer.

Выводы

Получен навык работы с ООП. Изучены особенности реализации данной методологии программирования в языке Python.

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")
          self.gender = gender
          if not(type(age) == type(1) and age > 0):
               raise ValueError("Invalid value")
          self.age = age
          if not(type(height) == type(1) and height > 0):
               raise ValueError("Invalid value")
          self.height = height
          if not(type(weight) == type(1) and weight > 0):
               raise ValueError("Invalid value")
          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(type(forces) == type(1) and forces > 0):
               raise ValueError("Invalid value")
          self.forces = forces
          if not(type(physical_damage) == type(1) and physical_damage >
     0):
               raise ValueError("Invalid value")
          self.physical_damage = physical_damage
          if not(type(armor) == type(1) and armor > 0):
               raise ValueError("Invalid value")
          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):
          if self.physical_damage == other.physical_damage and
     self.forces == other.forces and self.armor == other.armor:
               return True
          return False

class Magician(Character): #Наследуется от класса Character
     def __init__(self, gender, age, height, weight, mana, magic_damage):
          super().__init__(gender, age, height, weight)
          if not(type(mana) == type(1) and mana > 0):
               raise ValueError("Invalid value")
          self.mana = mana
          if not(type(magic_damage) == type(1) and magic_damage > 0):
               raise ValueError("Invalid value")
          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.magic_damage * self.mana

class Archer(Character):
#Наследуется от класса Character
      def   __init__(self,   gender,   age,   height,   weight,   forces,
physical_damage, attack_range):
            super().__init__(gender, age, height, weight)
            if not(type(forces) == type(1) and forces > 0):
                  raise ValueError("Invalid value")
            self.forces = forces
            if not(type(physical_damage) == type(1) and physical_damage >
      0):
                  raise ValueError("Invalid value")
            self.physical_damage = physical_damage
            if not(type(attack_range) == type(1) and attack_range > 0):
                  raise ValueError("Invalid value")
            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):
            if    self.physical_damage    ==    other.physical_damage    and
      self.forces    ==    other.forces    and    self.attack_range    ==
      other.attack_range:
                  return True
            return False

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

      def append(self, p_object):
            if not isinstance(p_object, Warrior):
                  raise TypeError(f"Invalid type {type(p_object)}")
            super().append(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 x in iterable:
                  if isinstance(x, Magician): super().append(x)
      def print_damage(self):
            print(sum([x.magic_damage for x in self]))
```

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

      def append(self, p_object):
            if not isinstance(p_object, Archer):
                  raise TypeError(f"Invalid type {type(p_object)}")
            super().append(p_object)

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