Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _*ИУК «Информатика и управление»*_____

КАФЕДРА __*ИУК2 "Информационные системы и сети"*

ЛАБОРАТОРНАЯ РАБОТА №3

«ОБЪЕКТНО-ОРИИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА PYTHON »

ДИСЦИПЛИНА: «Перспективные языки программирования»

Выполнил: студент гр. ИУК4-33Б _____ (___Сарпян Н.А.___)
(Подпись) (Ф.И.О.)

Проверил: _____ (___Осипова О.В.___)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2024

Цель: Целью выполнения лабораторной работы является формирование практических навыков процедурного программирования, разработки и отладки программ, овладение методами и средствами разработки и оформления технической документации.

Задачи:

1. Изучить особенности создания классов;
2. Научиться создавать экземпляры классов;
3. Изучить типовые алгоритмы решения задач с

   использованием принципов объектно-ориентированного программирования.

Задание:

Задача 1. Создайте класс Parent, задайте параметр grow определите метод colorEye, выводящий предложение о зелёном цвете глаз, метод changeGrow – вычисляющий изменение роста, метод printGrow – печатающий величину роста. Создайте два класса-наследника Masha и Peter, задайте параметр роста для них. В одном из данных классов переопределите метод colorEye, заменив цвет глаз на карий. Задайте параметры изменения роста для имеющихся классов. Выведите сообщение о росте и цвете глаз каждого класса.

Задача 2. Создать абстрактный класс, описывающие возможности игровых персонажей (бег, плавание, прыжки). Создать базовый класс персонажа для игры и два унаследованных класса персонажа, которые используют реализованные методы абстрактного класса для определения своих возможностей.

Задача 3. Создать класс Holiday и описать в нем название праздника, имя приглашенного и количество персон в этом приглашении. Описать метод family, определяющий может ли вся семья приглашенного (количество задается с клавиатуры) посетить данный праздник

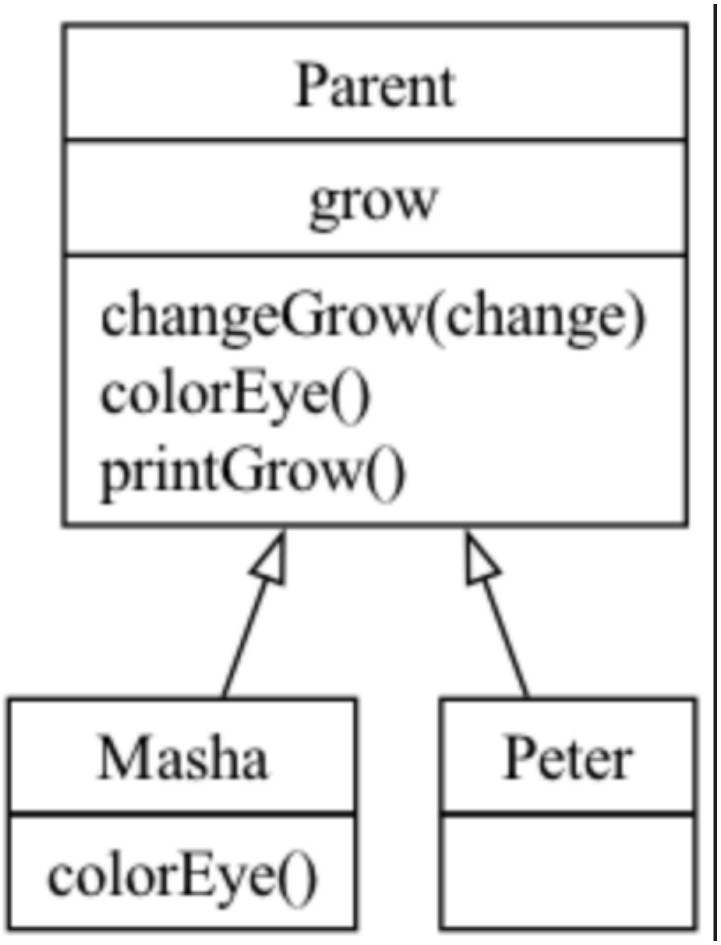
Задача 4. Для описания всех людей, находящихся в магазине, необходимо выделить подмножество работников и подмножество посетителей. У каждого человека есть полное имя, у сотрудников магазина имеет значение должность, а у посетителей возраст. Необходимо сгенерировать список людей в магазине и вызовом виртуального метода из абстрактного класса напечатать для сотрудников фамилию и должность, а для посетителей имя и возраст.

Листинг программы задания 1:

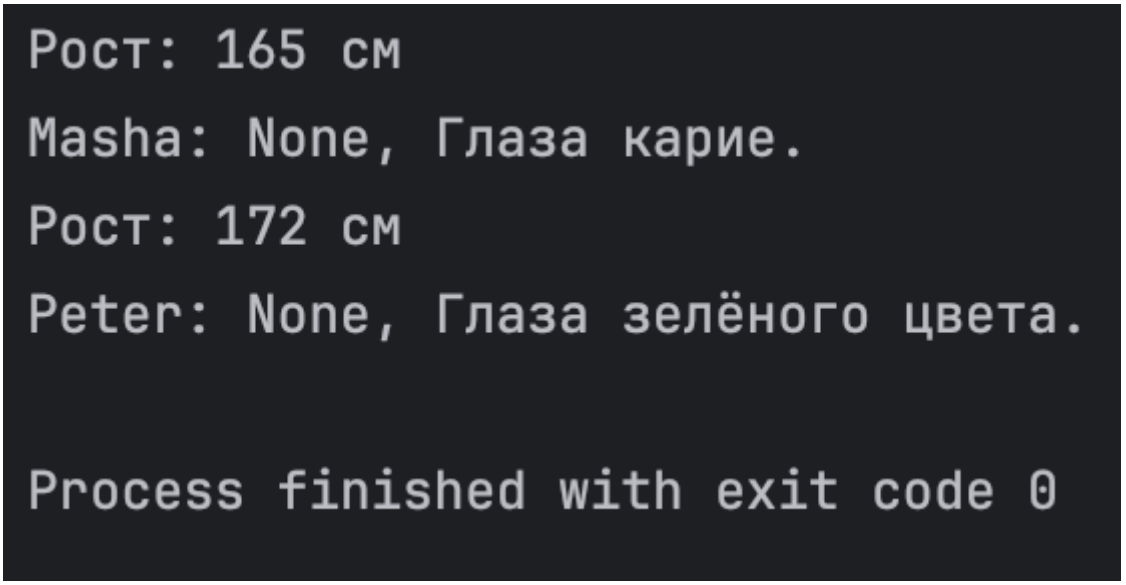
```
class Parent:
    def __init__(self, grow):
        self.grow = grow

    def colorEye(self):
        return "Глаза зелёного цвета."

    def changeGrow(self, change):
        self.grow += change

    def printGrow(self):
```

```
        print(f"Рост: {self.grow} см")


class Masha(Parent):
    def __init__(self, grow):
        super().__init__(grow)

    def colorEye(self):
        return "Глаза карие."


class Peter(Parent):
    def __init__(self, grow):
        super().__init__(grow)


masha = Masha(grow=160)
peter = Peter(grow=175)

masha.changeGrow(5)
peter.changeGrow(-3)

print(f"Masha: {masha.printGrow()}, {masha.colorEye()}")
print(f"Peter: {peter.printGrow()}, {peter.colorEye()}")
```

UML диаграмма:



Результат выполнения программы

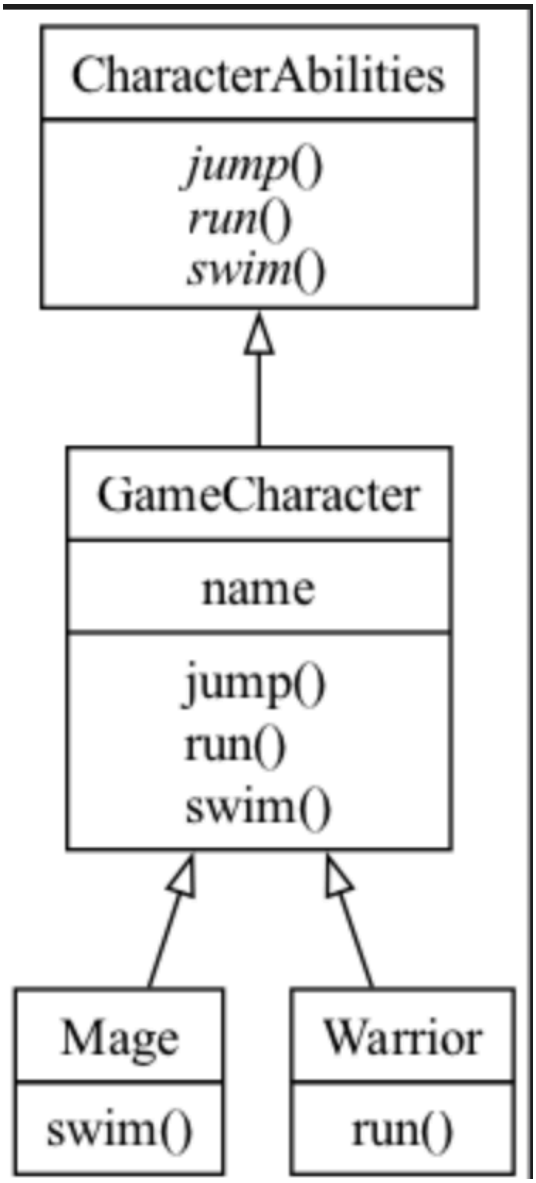


Листинг программы задания 2.

```
from abc import ABC, abstractmethod


class CharacterAbilities(ABC):
    @abstractmethod
    def run(self):
        pass

    @abstractmethod
    def swim(self):
        pass

    @abstractmethod
    def jump(self):
        pass


class GameCharacter(CharacterAbilities):
    def __init__(self, name):
        self.name = name

    def run(self):
        return f"{self.name} бегает быстро."

    def swim(self):
        return f"{self.name} плавает хорошо."

    def jump(self):
        return f"{self.name} прыгает высоко."


class Warrior(GameCharacter):
    def __init__(self, name):
        super().__init__(name)

    def run(self):
        return f"{self.name} бегает с тяжёлой бронёй."
```

```
class Mage(GameCharacter):
    def __init__(self, name):
        super().__init__(name)

    def swim(self):
        return f"{self.name} использует магию для плавания."


warrior = Warrior(name="Воин")
mage = Mage(name="Маг")

print(warrior.run())
print(warrior.swim())
print(warrior.jump())

print(mage.run())
print(mage.swim())
print(mage.jump())
```

UML диаграмма:



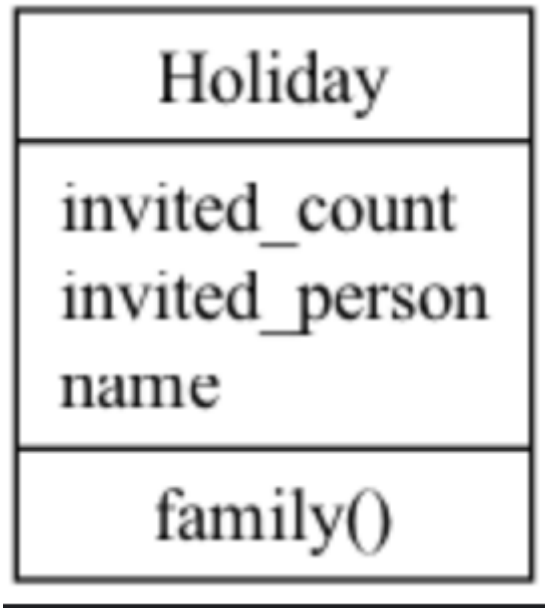
Результат выполнения программы

```
Воин бегает с тяжёлой бронёй.
Воин плавает хорошо.
Воин прыгает высоко.
Маг бегает быстро.
Маг использует магию для плавания.
Маг прыгает высоко.


Process finished with exit code 0
```

Листинг программы задания 3

```
class Holiday:
    def __init__(self, name, invited_person, invited_count):
        self.name = name
        self.invited_person = invited_person
        self.invited_count = invited_count

    def family(self):
        family_count = int(input("Введите количество членов семьи: "))
        if family_count <= self.invited_count:
              return f"Вся семья {self.invited_person} может посетить праздник
{self.name}."
        else:
               return f"К сожалению, вся семья {self.invited_person} не сможет
посетить праздник {self.name}."

holiday_name = input("Введите название праздника: ")
invited_person = input("Введите имя приглашенного: ")
invited_count = int(input("Введите количество персон в приглашении: "))

holiday = Holiday(holiday_name, invited_person, invited_count)
result = holiday.family()
print(result)
```

UML диаграмма:



Результат выполнения программы

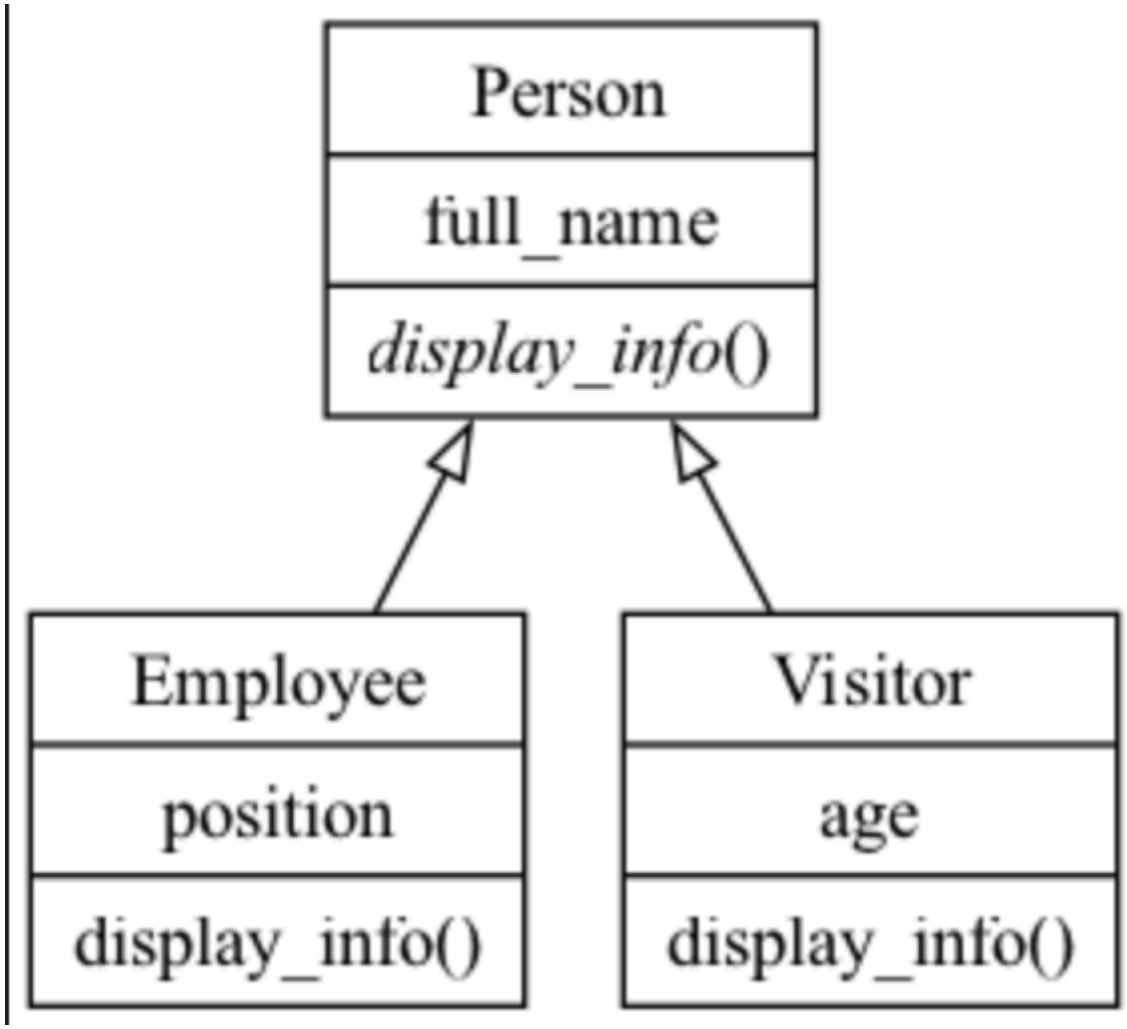
```
Введите название праздника: Новый год
Введите имя приглашенного: Иисус Христос
Введите количество персон в приглашении: 12
Введите количество членов семьи: 10
Вся семья Иисус Христос может посетить праздник Новый год.

Process finished with exit code 0
```

Листинг программы задания 4

```
from abc import ABC, abstractmethod


class Person(ABC):
    def __init__(self, full_name):
        self.full_name = full_name

    @abstractmethod
    def display_info(self):
        pass


class Employee(Person):
    def __init__(self, full_name, position):
        super().__init__(full_name)
        self.position = position
```

```
    def display_info(self):
        last_name = self.full_name.split()[-1]
        return f"Сотрудник: {last_name}, Должность: {self.position}"


class Visitor(Person):
    def __init__(self, full_name, age):
        super().__init__(full_name)
        self.age = age

    def display_info(self):
        return f"Посетитель: {self.full_name}, Возраст: {self.age}"


people_in_store = [
    Employee("Иванов Иван Иванович", "Менеджер"),
    Visitor("Петров Петр Петрович", 30),
    Employee("Сидоров Сидор Сидорович", "Кассир"),
    Visitor("Кузнецова Анна Сергеевна", 25),
]

for person in people_in_store:
    print(person.display_info())
```

UML диаграмма:



Результат выполнения программы

```
Сотрудник: Иванович, Должность: Менеджер
Посетитель: Петров Петр Петрович, Возраст: 30
Сотрудник: Сидорович, Должность: Кассир
Посетитель: Кузнецова Анна Сергеевна, Возраст: 25

Process finished with exit code 0
```

Вывод: в ходе выполнения работы были приобретены практические навыки необходимые для разработки задач, решение которых предполагает использование классов и парадигмы ООП средствами языка Python.