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 ИУК4 «Программное обеспечение ЭВМ, информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №3

«Объектно-Ориентированное Программирование на Python»

ДИСЦИПЛИНА: «Перспективные языки программирования»

Выполнил: студент гр. ИУК4-31Б _____ (Суриков Н. С.)
(подпись) (Ф.И.О.)

Проверил: _____ (Осипова О. В.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2024

Цель: формирование практических навыков объектно-ориентированного 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.

Вариант 5

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.

Листинг программы 1:

parent.py

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

    def colorEye(self):
        return "У меня зеленые глаза."

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

    def printGrow(self):
        print(f"Мой рост: {self.grow} см.")

```

masha.py

```python
from parent import Parent


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

    def colorEye(self):
        return "У меня карие глаза."

```

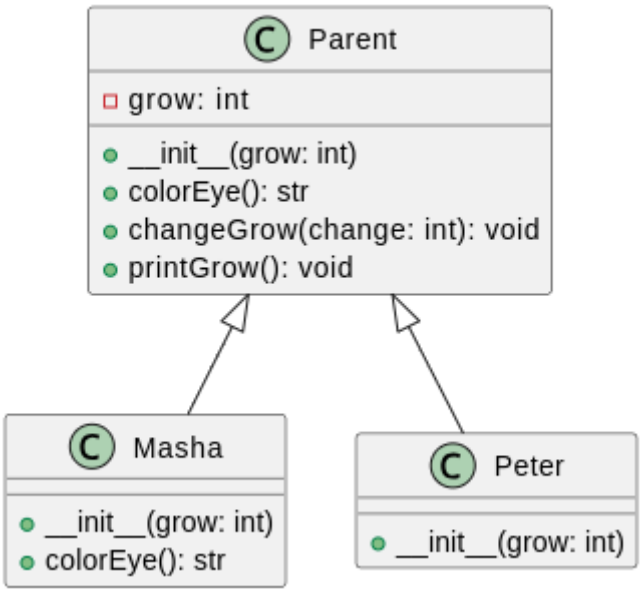
peter.py

```python
from parent import Parent


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

```

main.py

```python
from masha import Masha
from peter import Peter

masha = Masha(grow=160)
peter = Peter(grow=170)

masha.changeGrow(change=5)
peter.changeGrow(change=3)

print(f"{'-'*5} Маша {'-'*5}")
print(masha.colorEye())
masha.printGrow()

print(f"{'-'*5} Петя {'-'*5}")
print(peter.colorEye())
peter.printGrow()

```

UML диаграмма программы 1:



Результат работы программы 1:

```
----- Маша -----
У меня карие глаза.
Мой рост: 165 см.
----- Петя -----
У меня зеленые глаза.
Мой рост: 173 см.
```

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.

Листинг программы 2:

character_abilities.py

```
1  from abc import ABC, abstractmethod
2
3
4  class CharacterAbilities(ABC):
5      @abstractmethod
6      def run(self):
7          pass
8
```

```
    @abstractmethod
    def swim(self):
        pass

    @abstractmethod
    def jump(self):
        pass

```

game_character.py

```
from character_abilities import CharacterAbilities


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

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

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

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

```

mage.py

```
from game_character import GameCharacter


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

    def run(self):
        return f"{self.name} бегает, используя магию для ускорения."

    def swim(self):
        return f"{self.name} может плавать, используя магические заклинания."

    def jump(self):
        return f"{self.name} делает волшебный прыжок."

```

warrior.py

```
from game_character import GameCharacter


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

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

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

    def jump(self):
        return f"{self.name} делает мощный прыжок."

```

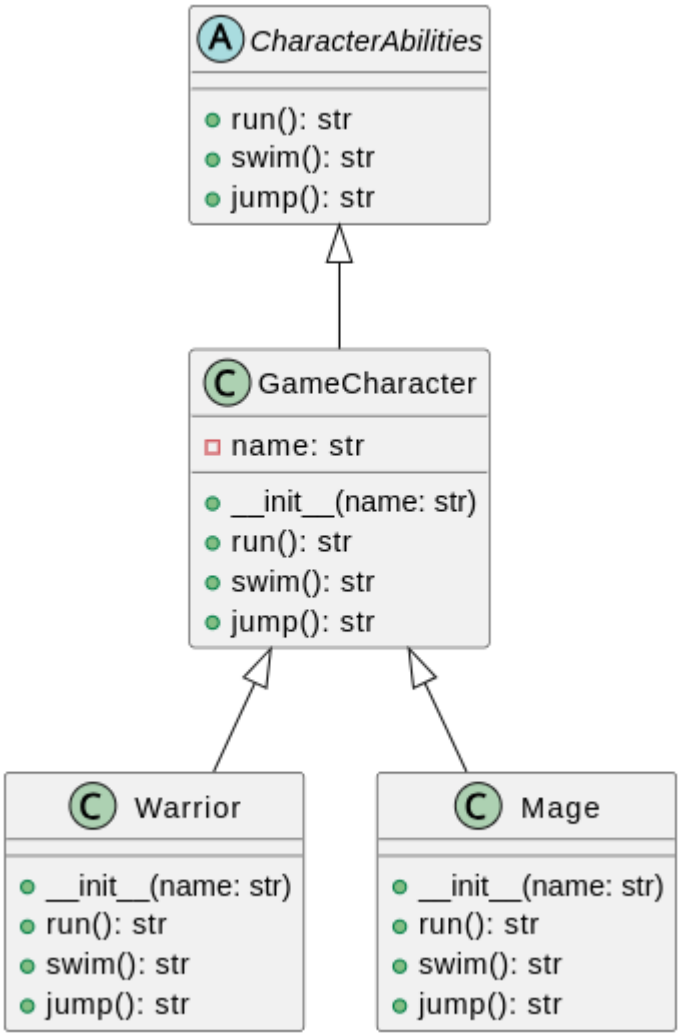
main.py

```
from warrior import Warrior
from mage import Mage

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

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

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

```

Результат работы программы 2:

```
Воин быстро бегает с оружием.
Воин не очень хорошо плавает.
Воин делает мощный прыжок.
Маг бегает, используя магию для ускорения.
Маг может плавать, используя магические заклинания.
Маг делает волшебный прыжок.
```

Блок схема программы 2:



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.

Листинг программы 3:

holiday.py

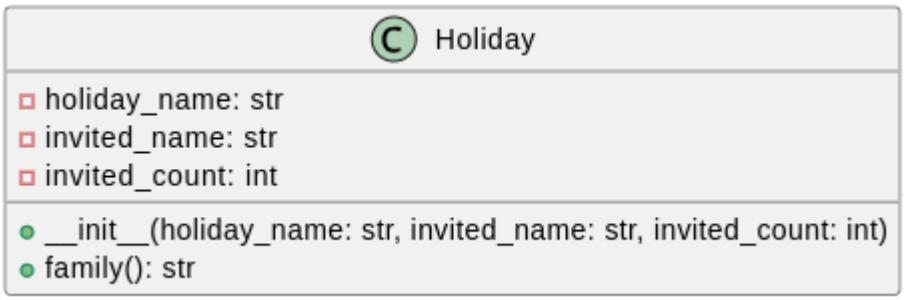
```
1  class Holiday:
2      def __init__(self, holiday_name, invited_name, invited_count):
3          self.holiday_name = holiday_name
4          self.invited_name = invited_name
5          self.invited_count = invited_count
6
7      def family(self):
```

```
        try:
            family_count = int(input("Введите количество членов семьи: "))
            if family_count <= self.invited_count:
                return f"{self.invited_name} может пригласить всю \
                            семью на праздник {self.holiday_name}."
            else:
                return f"{self.invited_name} не может пригласить \
                            всю семью на праздник {self.holiday_name}."
        except ValueError:
            return "Пожалуйста, введите корректное число."

```

main.py

```
from holiday import Holiday

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

holiday = Holiday(holiday_name, invited_name, invited_count)

result = holiday.family()
print(result)

```

Блок схема программы 3:



Результат работы программы 3:

```
Введите название праздника: Рождество
Введите имя приглашенного: Никита
Введите количество персон в приглашении: 5
Введите количество членов семьи: 7
Никита не может пригласить всю семью на праздник Рождество.

Введите название праздника: Свадьба
Введите имя приглашенного: Никита
Введите количество персон в приглашении: 10
Введите количество членов семьи: 5
Никита может пригласить всю семью на праздник Свадьба.
```

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 4:

person.py

```
from abc import ABC, abstractmethod


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

    @abstractmethod
    def display_info(self):
        pass

```

employee.py

```
from person import Person


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}"

```

visitor.py

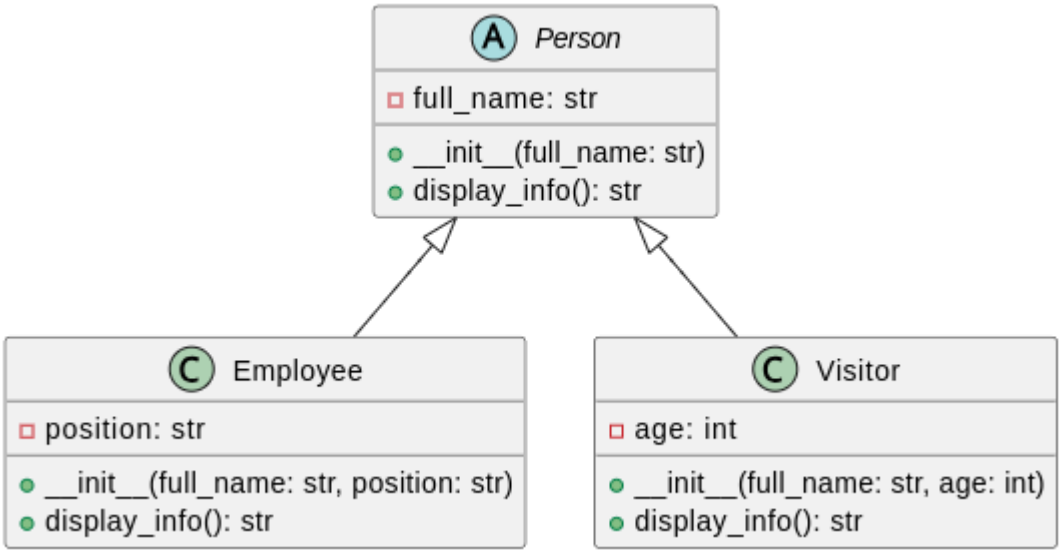
```
from person import Person


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

```
8
9       def display_info(self):
10          return f"Посетитель: {self.full_name}, Возраст: {self.age}"
11
```

main.py

```
1  from employee import Employee
2  from visitor import Visitor
3
4
5  def main():
6      people_in_store = [
7          Employee("Иванов Иван Иванович", "Менеджер"),
8          Visitor("Петрова Анна Сергеевна", 30),
9          Employee("Сидоров Алексей Владимирович", "Кассир"),
10         Visitor("Кузнецова Мария Петровна", 25),
11     ]
12
13     for person in people_in_store:
14         print(person.display_info())
15
16
17 if __name__ == "__main__":
18     main()
19
```

Блок схема программы 4:



Результат работы программы 4:

```
Сотрудник: Иванович, Должность: Менеджер
Посетитель: Петрова Анна Сергеевна, Возраст: 30
Сотрудник: Владимирович, Должность: Кассир
Посетитель: Кузнецова Мария Петровна, Возраст: 25
```

Вывод: в ходе работы были сформированы практические навыки объектно-ориентированного программирования, разработки и отладки программ, были освоены методы и средства разработки и оформления технической документации.