|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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: |

**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:*

*Cоздайте 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

1 class Parent:

2 def \_\_init\_\_(self, grow):

3 self.grow = grow

4

5 def colorEye(self):

6 return "У меня зеленые глаза."

7

8 def changeGrow(self, change):

9 self.grow += change

10

11 def printGrow(self):

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

13

masha.py

1 from parent import Parent

2

3

4 class Masha(Parent):

5 def \_\_init\_\_(self, grow):

6 super().\_\_init\_\_(grow)

7

8 def colorEye(self):

9 return "У меня карие глаза."

10

peter.py

1 from parent import Parent

2

3

4 class Peter(Parent):

5 def \_\_init\_\_(self, grow):

6 super().\_\_init\_\_(grow)

7

main.py

1 from masha import Masha

2 from peter import Peter

3

4 masha = Masha(grow=160)

5 peter = Peter(grow=170)

6

7 masha.changeGrow(change=5)

8 peter.changeGrow(change=3)

9

10 print(f"{'-'\*5} Маша {'-'\*5}")

11 print(masha.colorEye())

12 masha.printGrow()

13

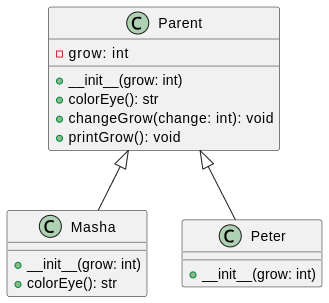
14 print(f"{'-'\*5} Петя {'-'\*5}")

15 print(peter.colorEye())

16 peter.printGrow()

17

**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

9 @abstractmethod

10 def swim(self):

11 pass

12

13 @abstractmethod

14 def jump(self):

15 pass

16

game\_character.py

1 from character\_abilities import CharacterAbilities

2

3

4 class GameCharacter(CharacterAbilities):

5 def \_\_init\_\_(self, name):

6 self.name = name

7

8 def run(self):

9 return f"{self.name} бегает."

10

11 def swim(self):

12 return f"{self.name} плавает."

13

14 def jump(self):

15 return f"{self.name} прыгает."

16

mage.py

1 from game\_character import GameCharacter

2

3

4 class Mage(GameCharacter):

5 def \_\_init\_\_(self, name):

6 super().\_\_init\_\_(name)

7

8 def run(self):

9 return f"{self.name} бегает, используя магию для ускорения."

10

11 def swim(self):

12 return f"{self.name} может плавать, используя магические заклинания."

13

14 def jump(self):

15 return f"{self.name} делает волшебный прыжок."

16

warrior.py

1 from game\_character import GameCharacter

2

3

4 class Warrior(GameCharacter):

5 def \_\_init\_\_(self, name):

6 super().\_\_init\_\_(name)

7

8 def run(self):

9 return f"{self.name} быстро бегает с оружием."

10

11 def swim(self):

12 return f"{self.name} не очень хорошо плавает."

13

14 def jump(self):

15 return f"{self.name} делает мощный прыжок."

16

main.py

1 from warrior import Warrior

2 from mage import Mage

3

4 warrior = Warrior("Воин")

5 mage = Mage("Маг")

6

7 print(warrior.run())

8 print(warrior.swim())

9 print(warrior.jump())

10

11 print(mage.run())

12 print(mage.swim())

13 print(mage.jump())

14

**Результат работы программы 2:**

Воин быстро бегает с оружием.

Воин не очень хорошо плавает.

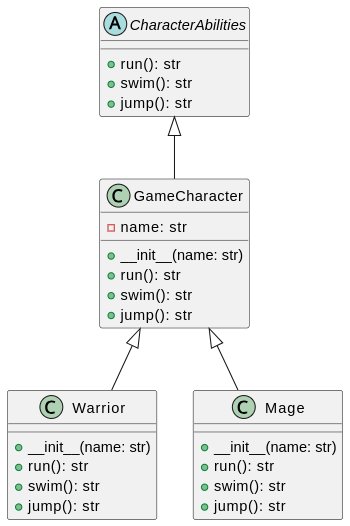
Воин делает мощный прыжок.

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):

8 try:

9 family\_count = int(input("Введите количество членов семьи: "))

10 if family\_count <= self.invited\_count:

11 return f"{self.invited\_name} может пригласить всю \

12 семью на праздник {self.holiday\_name}."

13 else:

14 return f"{self.invited\_name} не может пригласить \

15 всю семью на праздник {self.holiday\_name}."

16 except ValueError:

17 return "Пожалуйста, введите корректное число."

18

main.py

1 from holiday import Holiday

2

3 holiday\_name = input("Введите название праздника: ")

4 invited\_name = input("Введите имя приглашенного: ")

5 invited\_count = int(input("Введите количество персон в приглашении: "))

6

7 holiday = Holiday(holiday\_name, invited\_name, invited\_count)

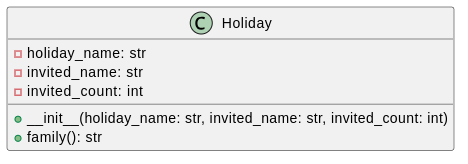
8

9 result = holiday.family()

10 print(result)

11

**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

1 from abc import ABC, abstractmethod

2

3

4 class Person(ABC):

5 def \_\_init\_\_(self, full\_name):

6 self.full\_name = full\_name

7

8 @abstractmethod

9 def display\_info(self):

10 pass

11

employee.py

1 from person import Person

2

3

4 class Employee(Person):

5 def \_\_init\_\_(self, full\_name, position):

6 super().\_\_init\_\_(full\_name)

7 self.position = position

8

9 def display\_info(self):

10 last\_name = self.full\_name.split()[-1]

11 return f"Сотрудник: {last\_name}, Должность: {self.position}"

12

visitor.py

1 from person import Person

2

3

4 class Visitor(Person):

5 def \_\_init\_\_(self, full\_name, age):

6 super().\_\_init\_\_(full\_name)

7 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

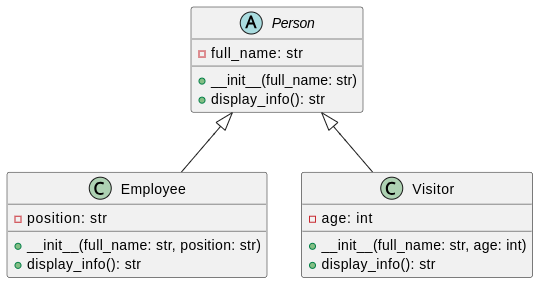
16

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.