ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ РЕСПУБЛИКИ ХАКАСИЯ  
«ХАКАССКИЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

Специальность 09.02.07

«Информационные системы и программирование»

ТЕМА: Наследование и коллекции в Python

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Студент гр. | ИС(ТП)-31 |  |  | | | |  |  | Соломатин А.Д. | | |
|  |  |  | *подпись* | | | |  |  | *Фамилия, И.О.* | | |
| Руководитель |  |  |  |  |  |  | | | |  | Брюханова И.Н. |
|  | *оценка* |  | *дата* |  |  | *подпись* | | | |  | *Фамилия, И.О.* |

Абакан, 2024

# Практическая работа №\_2\_:

**Цель работы:**

**Задание №1**

1.Постановка задачи.

Реализуйте иерархию классов из не менее 3-х классов по

предметной области вашего варианта. Помимо системы,

предоставленной в предметной области – реализуйте управляющий

класс, который будет хранить коллекцию всех элементов и иметь

методы добавления, удаления из коллекции и поиска в ней нужного

элемента. Все исключительные ситуации должны быть обработаны.

Протестируйте созданные классы.

Например: дана система управления игровой площадкой. Вы

создаёте иерархию, где базовый класс – Игра, а её наследники

КонсольнаяИгра, ПКИгра и МобильнаяИгра. Для каждого класса

создаются атрибуты и специальные методы (если требуется).

Далее создаётся управляющий класс ИгроваяПлощадка, которая

имеет атрибут – список всех игр. Управляющий класс имеет

методы: ДобавитьИгру, УдалитьИгру, НайтиИгру.

10. Система управления медицинским учреждением;

2.Листинг программы:

class МедицинскийРаботник:

def \_\_init\_\_(self, имя, специализация):

self.имя = имя

self.специализация = специализация

def get\_info(self):

return f"Имя: {self.имя}, Специализация: {self.специализация}"

class Врач(МедицинскийРаботник):

def \_\_init\_\_(self, имя, специализация, стаж):

super().\_\_init\_\_(имя, специализация)

try:

self.стаж = int(стаж)

if self.стаж < 0:

raise ValueError("Стаж не может быть отрицательным.")

except ValueError as e:

print(f"Ошибка ввода стажа: {e}")

return

def get\_info(self):

return f"{super().get\_info()}, Стаж: {self.стаж} лет"

class Медсестра(МедицинскийРаботник):

def \_\_init\_\_(self, имя, специализация, отделение):

super().\_\_init\_\_(имя, специализация)

self.отделение = отделение

def get\_info(self):

return f"{super().get\_info()}, Отделение: {self.отделение}"

class МедицинскоеУчреждение:

def \_\_init\_\_(self):

self.работники = []

def добавить\_работника(self):

while True:

тип\_работника = input("Введите тип работника (Врач/Медсестра): ")

if тип\_работника.lower() == "врач":

имя = input("Введите имя врача: ")

специализация = input("Введите специализацию врача: ")

стаж = input("Введите стаж врача: ")

try:

врач = Врач(имя, специализация, стаж)

self.работники.append(врач)

print("Врач добавлен!")

break

except ValueError:

print("Ошибка ввода данных врача.")

elif тип\_работника.lower() == "медсестра":

имя = input("Введите имя медсестры: ")

специализация = input("Введите специализацию медсестры: ")

отделение = input("Введите отделение медсестры: ")

медсестра = Медсестра(имя, специализация, отделение)

self.работники.append(медсестра)

print("Медсестра добавлена!")

break

else:

print("Неверный тип работника.")

def get\_info\_о\_работнике(self, имя):

for работник in self.работники:

if работник.имя == имя:

return работник.get\_info()

return "Работник не найден"

учреждение = МедицинскоеУчреждение()

учреждение.добавить\_работника()

учреждение.добавить\_работника()

try:

print(учреждение.get\_info\_о\_работнике("Иван Иванов"))

print(учреждение.get\_info\_о\_работнике("Мария Петрова"))

print(учреждение.get\_info\_о\_работнике("Петр Сидоров"))

except Exception as e:

print(f"Ошибка: {e}")

3.Результаты



**Задание №2**

1.Постановка задачи.

Создайте иерархию классов-животных с базовым классом

Animal, который имеет базовые атрибуты: имя и возраст и метод say

(Говори). Создайте ещё один базовый класс BreedAnimal (Породистое

Животное) с атрибутом breed – порода. Создайте классы Cat и Dog и

наследуйте их от BreedAnimal. Создайте классы Mouse и Rat и

наследуйте их от Animal. Переопределите метод Say для каждого

класса.

Создайте класс-коллекцию «Queue» (Очередь), который будет

имитировать очередь. У классы должны быть реализованы методы –

добавить, удалить и показать первого в очереди. Добавление в

очередь выполняется в конец списка, а удаление из очереди – в начало

списка.

Создайте наследника класса Queue – AnimalQueue, который

переопределяет метод добавления в очередь. На вход он принимает

только экземпляр типа Animal и его наследников. Очередь не должна

содержать экземпляры типа Mouse и Rat, поэтому при попытке

добавления в очередь экземпляров типа Mouse или Rat – программа

должна выдавать сообщение об ошибках.

2.Листинг программы:

class Animal:

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

self.name = name

self.age = age

def say(self):

print("Звук животного")

class BreedAnimal(Animal):

def \_\_init\_\_(self, name, age, breed):

super().\_\_init\_\_(name, age)

self.breed = breed

class Cat(BreedAnimal):

def say(self):

print("Мяу")

class Dog(BreedAnimal):

def say(self):

print("Гав")

class Mouse(Animal):

def say(self):

print("Писк")

class Rat(Animal):

def say(self):

print("Скрип")

class Queue:

def \_\_init\_\_(self):

self.queue = []

def add(self, item):

self.queue.append(item)

print(f"Животное {item.name} добавлено в очередь.")

def remove(self):

if self.queue:

removed = self.queue.pop(0)

print(f"Животное {removed.name} удалено из очереди.")

else:

print("Очередь пуста!")

def peek(self):

if self.queue:

print(f"Первое животное в очереди: {self.queue[0].name}")

else:

print("Очередь пуста!")

class AnimalQueue(Queue):

def add(self, item):

if isinstance(item, (Animal, BreedAnimal)) and not isinstance(item, (Mouse, Rat)):

super().add(item)

else:

print("В очередь можно добавлять только животных, кроме мышей и крыс")

queue = AnimalQueue()

while True:

print("\nВыберите действие:")

print("1. Добавить животное в очередь")

print("2. Удалить животное из очереди")

print("3. Показать первое животное в очереди")

print("4. Выход")

choice = input("Введите номер действия: ")

try:

if choice == "1":

name = input("Введите имя животного: ")

age = int(input("Введите возраст животного: "))

animal\_type = input("Введите тип животного (Cat, Dog, Mouse, Rat): ")

if animal\_type == "Cat":

breed = input("Введите породу кота: ")

animal = Cat(name, age, breed)

elif animal\_type == "Dog":

breed = input("Введите породу собаки: ")

animal = Dog(name, age, breed)

elif animal\_type == "Mouse":

animal = Mouse(name, age)

elif animal\_type == "Rat":

animal = Rat(name, age)

else:

raise ValueError("Неверный тип животного.")

queue.add(animal)

elif choice == "2":

queue.remove()

elif choice == "3":

queue.peek()

elif choice == "4":

break

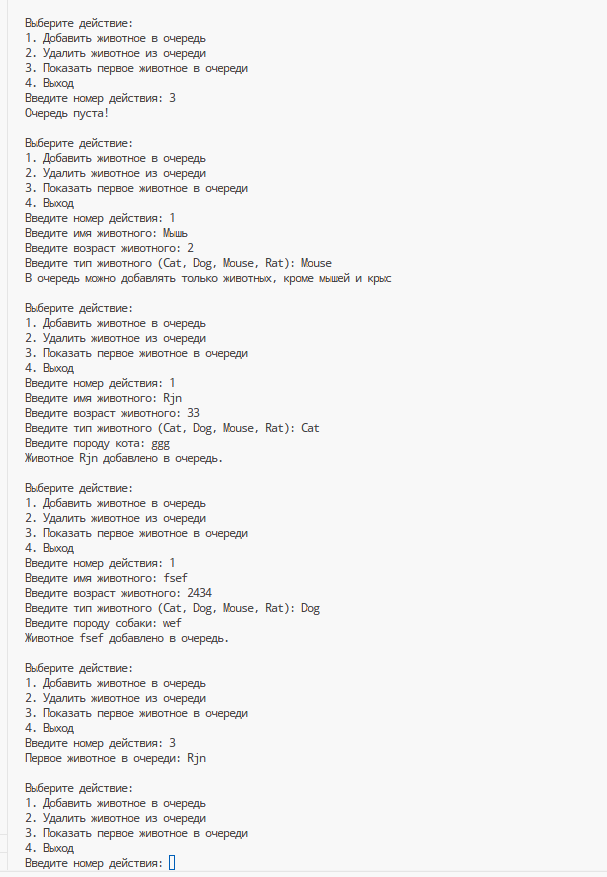
else:

raise ValueError("Неверный выбор.")

except ValueError as e:

print(f"Ошибка: {e}")

3.Результаты



**Задание №4**

1. Что такое наследование и как оно реализуется в Python?

Наследование - это механизм, позволяющий создавать новые классы (подклассы) на основе уже существующих классов (суперклассов), наследуя их свойства и методы.

class Animal:

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

self.name = name

def say(self):

print("Звук животного")

class Dog(Animal):

def say(self):

print("Гав")

dog = Dog("Рекс")

dog.say() # Вывод: Гав

2. Назовите три основных вида коллекций в Python

- Списки (list): Упорядоченные, изменяемые коллекции, хранящие элементы разных типов данных.

- Словари (dictionary): Неупорядоченные, изменяемые коллекции, хранящие пары ключ-значение.

- Кортежи (tuple): Упорядоченные, неизменяемые коллекции, хранящие элементы разных типов данных.

3. Что такое отношение «ассоциация»?

Ассоциация - это отношение между двумя классами, которое указывает на то, что объекты этих классов взаимодействуют друг с другом.

- Студент и Курс: Студент может быть записан на несколько курсов, а курс может иметь множество студентов.

4. Что такое отношение «композиция»?

Композиция - это форма сильного отношения «часть-целое», где объект одного класса (компонент) является частью объекта другого класса (целого) и его существование зависит от целого.

Пример:

- Автомобиль и Двигатель: Двигатель является частью автомобиля и не может существовать без него.

5. Что такое отношение «агрегация»?

Агрегация - это более слабая форма отношения «часть-целое», чем композиция. Объект одного класса (компонент) может существовать независимо от объекта другого класса (целого), но он является частью этого целого.

Пример:

- Команда и Игрок: Игрок может принадлежать к команде, но может также существовать и без нее.