МИНЕСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Лабораторная работа №7

**Работа с классами ч.3**

**по дисциплине**

**«Введение в информационные технологии»**

Выполнил: студент гр. БВТ2403

Хусейнов С. И.

Проверил:

Москва, 2024 г.

**Цель работы:** Разработать систему управления сотрудниками, демонстрирующую множественное наследование, инкапсуляцию и полиморфизм в Python. Система должна уметь обрабатывать различные типы сотрудников, включая менеджеров и технических специалистов, а также предоставлять возможность для расширения и добавления новых ролей.

**Оборудование:** компьютер с установленным программным обеспечением Python.

**Практика:**

## Задание

class Employee:

    def \_\_init\_\_(self, name, id) -> None:

        self.name = name

        self.id = id

    def get\_info(self):

        return {'Name': self.name, 'ID': self.id}

class Manager(Employee):

    def \_\_init\_\_(self, name, id, department) -> None:

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

        self.department = department

    def manage\_project(self):

        return "управление проектами"

    def get\_info(self):

        return super().get\_info() | {'Department': self.department}

class Technician(Employee):

    def \_\_init\_\_(self, name, id, spacialization) -> None:

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

        self.spacialization = spacialization

    def perform\_maintenance(self):

        return "выполнение технического обслуживания"

    def get\_info(self):

        return super().get\_info() | {"Spacialization": self.spacialization}

class TechManager(Manager, Technician):

    def \_\_init\_\_(self, name, id, department, spacialization) -> None:

        self.name = name

        self.id = id

        self.department = department

        self.spacialization = spacialization

        self.subordinates = []

    def add\_employer(self, \*coworkers):

        self.subordinates.extend(coworkers)

    def get\_team\_info(self):

        info = list()

        team = self.subordinates

        for i in range(len(team)):

            info.append(team[i].get\_info())

        return info

emp0 = Employee('Name\_0', '5234' )

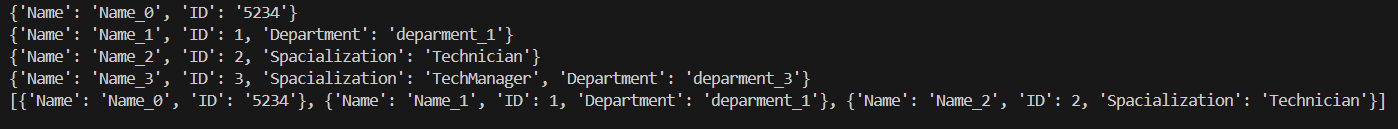
emp1 = Manager("Name\_1", 1, 'deparment\_1')

emp2 = Technician('Name\_2', 2, 'Technician')

emp3 = TechManager('Name\_3', 3, 'deparment\_3', 'TechManager')

emp3.add\_employer(emp0, emp1, emp2)

print(emp0.get\_info(), emp1.get\_info(), emp2.get\_info(), emp3.get\_info(), emp3.get\_team\_info(), sep='\n')



Создадим класс Employee с общими атрибутами, такими как **name** (имя), **id** (идентификационный номер) и методом **get\_info**(), который возвращает базовую информацию о сотруднике.

Создадим класс **Manager** с дополнительными атрибутами, такими как **department** (отдел) и методом, **manage\_project**(), символизирующим управление проектами.

Созададим класс **Technician** с уникальными атрибутами, такими как **specialization** (специализация), и методом **perform\_maintenance**(), означающим выполнение технического обслуживания.

Создадим класс **TechManager**, который наследует как Manager, так и Technician. Добавим метод **add\_employee(),** который позволяет **TechManager** добавлять сотрудников в список подчинённых, а также реализуем метод **get\_team\_info**(), который выводит информацию о всех подчинённых сотрудниках.

Создаем объекты каждого класса и демонстрируем их функциональность.

**Вывод:** Мы научились использовать множественное наследование, инкапсуляцию и полиморфизм в Python.