**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

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

Кафедра «Математическая Кибернетика и Информационные технологии»

Лабораторная работа №7

# Работа с классами ч.3

Выполнил: Студент группы

БВТ2402

Бадма-Халгаев Артен

Москва

2024

**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.

**Задание:**

Программа, содержащая несколько классов: второй и третий наследуются от первого и содержат дополнительные атрибуты и методы, четвертый наследуется от второго и третьего и содержит дополнительный метод.

**Ход работы:**

Код программы:

class Employee:  
 def \_\_init\_\_(self, name, id, \*\*kwargs):  
 self.name = name  
 self.id = id  
 super().\_\_init\_\_(\*\*kwargs)  
  
 def get\_info(self):  
 return "Имя: {}, ID: {}".format(self.name, self.id)  
  
  
class Manager(Employee):  
 def \_\_init\_\_(self, name, id, department, \*\*kwargs):  
 self.department = department  
 super().\_\_init\_\_(name=name, id=id, \*\*kwargs)  
  
 def get\_info(self):  
 return "Имя: {}, ID: {}, Отдел: {}".format(self.name, self.id, self.department)  
  
 def manage\_project(self):  
 return "Менеджер {} управляет проектом в отделе <<{}>>".format(self.name, self.department)  
  
  
class Technician(Employee):  
 def \_\_init\_\_(self, name, id, specialization, \*\*kwargs):  
 self.specialization = specialization  
 super().\_\_init\_\_(name=name, id=id, \*\*kwargs)  
  
 def get\_info(self):  
 return "Имя: {}, ID: {}, Специализация: {}".format(self.name, self.id, self.specialization)  
  
 def perform\_maintenance(self):  
 return "Техник {} осуществляет тех обслуживание в области <<{}>>".format(self.name, self.specialization)  
  
  
class TechManager(Manager, Technician):  
 def \_\_init\_\_(self, name, id, department, specialization):  
 super().\_\_init\_\_(name=name, id=id, department=department, specialization=specialization)  
 self.team = []  
  
 def get\_info(self):  
 return "Имя: {}, ID: {}, Отдел: {}, Специализация: {}".format(self.name, self.id, self.department, self.specialization)  
  
 def add\_employee(self, emp):  
 self.team.append(emp)  
  
 def get\_team\_info(self):  
 return "Команда:\n" + "\n".join([i.get\_info() for i in self.team])  
  
  
ivan = Employee("Иван", "101")  
anya = Manager("Анастасия", "102", "IT")  
pavel = Technician("Павел", "103", "Электроника")  
alex = TechManager("Алексей", "104", "Разработка", "Програмное обеспечение")  
  
print(ivan.get\_info())  
print(anya.get\_info())  
print(pavel.get\_info())  
print(alex.get\_info())  
  
print(anya.manage\_project())  
print(pavel.perform\_maintenance())  
print(alex.manage\_project())  
print(alex.perform\_maintenance())  
  
alex.add\_employee(ivan)  
alex.add\_employee(anya)  
alex.add\_employee(pavel)  
  
print(alex.get\_team\_info())

**Вывод**: Мы 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.