**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования**

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

Кафедра “Математическая кибернетика и информационные технологии”

ОТЧЕТ

по лабораторной работе №7

по дисциплине «Введение в информационные технологии»

Тема: «Работа с классами ч.3»

Выполнил: студент группы БВТ2505

Вакалюк А.А.

Проверил: Павликов. А.Е.

Москва, 2025

**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:**

1. Создать класс Employee с общими атрибутами, такими как name (имя), id (идентификационный номер) и методами, например, get_info(), который возвращает базовую информацию о сотруднике.

2. Создать класс Manager с дополнительными атрибутами, такими как department (отдел) и методами, например, manage_project(), символизирующим управление проектами.

3. Создать класс Technician с уникальными атрибутами, такими как specialization (специализация), и методами, например, perform_maintenance(), означающим выполнение технического обслуживания.

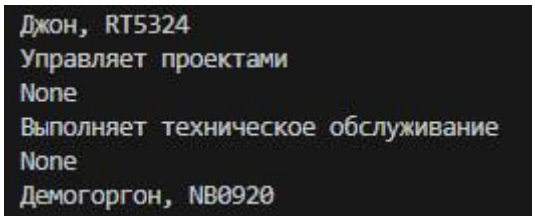
4. Создать класс TechManager, который наследует как Manager, так и Technician. Этот класс должен комбинировать управленческие способности и технические навыки, например, иметь методы для управления проектами и выполнения технического обслуживания.

5. Добавить метод add_employee(), который позволяет TechManager добавлять сотрудников в список подчинённых.

6. Реализовать метод get_team_info(), который выводит информацию о всех подчинённых сотрудниках.

7. Создать объекты каждого класса и демонстрируйте их функциональность.

**Скриншоты выполнения:**




**Исходный код программы:**

```
class Employee:
    def __init__(self, name, id):
        self.name = name
        self.id = id

    def get_info(self):
        return f'{self.name}, {self.id}'

class Manager(Employee):
    def __init__(self, name, id, department):
```

```
        Employee.__init__(self, name, id)
        self.department = department

    def get_info(self):
        return f'{super().get_info}, {self.department}'

    def manage_project(self):
        print('Управляет проектами')

class Technician(Employee):
    def __init__(self, name, id, specialization):
        Employee.__init__(self, name, id)
        self.specialization = specialization

    def get_info(self):
        return f'{super().get_info}, {self.specialization}'

    def perform_maintenance(self):
        print('Выполняет техническое обслуживание')

class TechManager(Manager, Technician):
    def __init__(self, name, id, department, specialization):
        Manager.__init__(self, name, id, department)
        Technician.__init__(self, name, id, specialization)
        self.team = []

    def get_info(self):
        return f'{Employee.get_info(self)}'

    def manage_project(self):
        print('Управляет проектами')

    def perform_maintenance(self):
        print('Выполняет техническое обслуживание')

    def add_employee(self, employee):
        self.team.append(employee)

    def get_team_info(self):
        return f'{self.team}'

emp = Employee('Джон', 'RT5324')
man = Manager('Эдвард', 'UI8347', 'IT-отдел')
tech = Technician('Стив', 'PL8439', 'QA-инженер')
TM = TechManager('Демогоргон', 'NB0920', 'Кибербезопасность', 'Инженер по
безопасности')

print(emp.get_info())
```

```
print(man.manage_project())
print(tech.perform_maintenance())
print(TM.get_info())
TM.add_employee(man)
TM.add_employee(tech)
print(TM.get_team_info())
```

## Заключение

**Вывод по лабораторной работе:**

В ходе выполнения лабораторной работы была реализована иерархия классов на основе принципов объектно-ориентированного программирования (ООП) — наследования, инкапсуляции и полиморфизма. Базовый класс `Employee` содержит общие атрибуты и метод для получения информации о сотруднике. От него наследуются два специализированных класса — `Manager` и `Technician`, каждый из которых расширяет функциональность базового класса в соответствии со своей ролью.