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

Хардиков Владислав Дмитриевич

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:

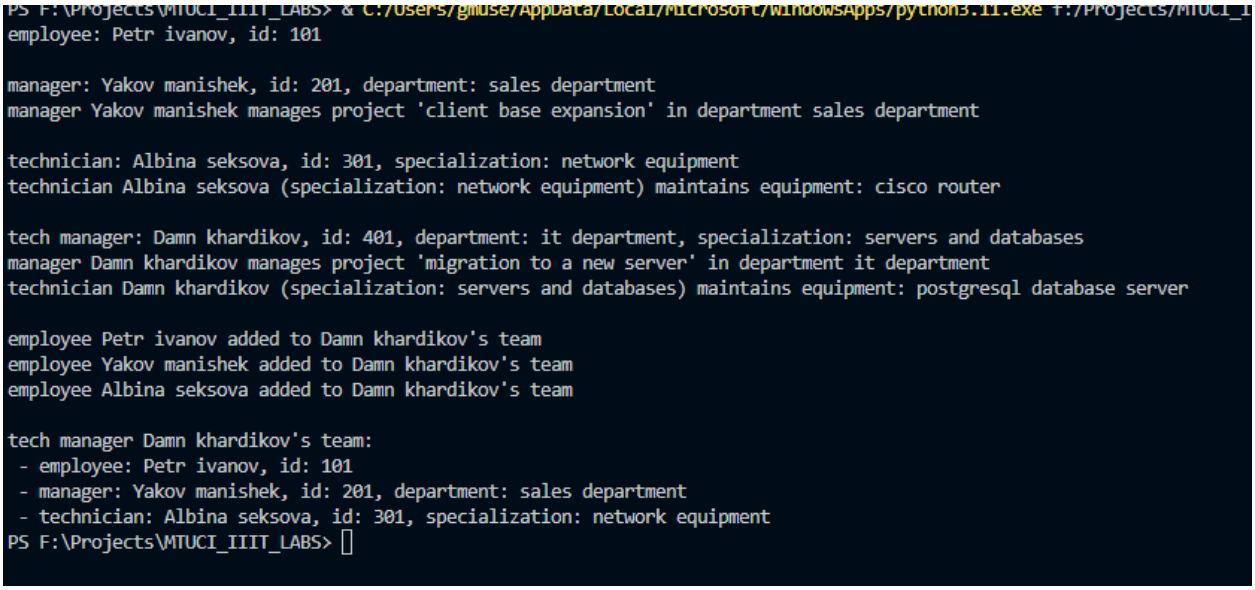

Рисунок 1. Результат выполнения программы.

Исходный код программы:

```
class Employee:
    def __init__(self, name: str, emp_id: int):
        self.name = name
        self.emp_id = emp_id

    def get_info(self) -> str:
        return f"employee: {self.name}, id: {self.emp_id}"

class Manager(Employee):
    def __init__(self, name: str, emp_id: int, department: str):
        Employee.__init__(self, name, emp_id)
        self.department = department

    def manage_project(self, project_name: str) -> str:
        return f"manager {self.name} manages project '{project_name}' in department
{self.department}"

    def get_info(self) -> str:
        return f"manager: {self.name}, id: {self.emp_id}, department: {self.department}"

class Technician(Employee):
    def __init__(self, name: str, emp_id: int, specialization: str):
        Employee.__init__(self, name, emp_id)
        self.specialization = specialization

    def perform_maintenance(self, equipment: str) -> str:
        return (f"technician {self.name} (specialization: {self.specialization}) "
                f"maintains equipment: {equipment}")

    def get_info(self) -> str:
        return f"technician: {self.name}, id: {self.emp_id}, specialization: {self.specialization}"

class TechManager(Manager, Technician):
    def __init__(self, name: str, emp_id: int, department: str, specialization: str):
        Manager.__init__(self, name, emp_id, department)
        Technician.__init__(self, name, emp_id, specialization)
        self._team = []  # list of subordinate employees

    def add_employee(self, employee: Employee) -> str:
        self._team.append(employee)
        return f"employee {employee.name} added to {self.name}'s team"

    def get_team_info(self) -> str:
        if not self._team:
            return f"{self.name} has no subordinates yet."

        result = [f"tech manager {self.name}'s team:"]
```

```
        for emp in self._team:
            result.append(" - " + emp.get_info())
        return "\n".join(result)

    # we can override get_info one more time
    def get_info(self) -> str:
        return (f"tech manager: {self.name}, id: {self.emp_id}, department: {self.department}, "
                f"specialization: {self.specialization}")

# demonstration of how it works
if __name__ == "__main__":
    # 1. regular employee
    e1 = Employee("Petr ivanov", 101)
    print(e1.get_info())
    print()

    # 2. manager
    m1 = Manager("Yakov manishek", 201, "sales department")
    print(m1.get_info())
    print(m1.manage_project("client base expansion"))
    print()

    # 3. technician
    t1 = Technician("Albina seksova", 301, "network equipment")
    print(t1.get_info())
    print(t1.perform_maintenance("cisco router"))
    print()

    # 4. tech manager (multiple inheritance)
    tm1 = TechManager("Damn khardikov", 401, "it department", "servers and databases")
    print(tm1.get_info())
    print(tm1.manage_project("migration to a new server"))
    print(tm1.perform_maintenance("postgresql database server"))
    print()

    # 5. adding employees
    print(tm1.add_employee(e1))
    print(tm1.add_employee(m1))
    print(tm1.add_employee(t1))
    print()

    # 6. printing team info
    print(tm1.get_team_info())
```

Заключение

В этой лабораторной работе были созданы классы сотрудников: обычный сотрудник, менеджер, техник и технический менеджер. У каждого класса были свои атрибуты и методы. Также был реализован класс TechManager, который наследует возможности сразу от двух классов (Manager и Technician).

Были добавлены методы для управления сотрудниками, добавления подчинённых и вывода информации о команде. Все классы были протестированы с помощью созданных объектов.

В итоге работа помогла научиться использовать наследование, в том числе множественное, а также создавать более сложные структуры классов и связывать их между собой.