МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,
СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Факультет «Информационные технологии»

Кафедра «Математическая кибернетика и информационные технологии»

Дисциплина: «Введение в информационные технологии»

Лабораторная работа №7

Работа с классами (часть 3)

Выполнил:

Студент группы БВТ2402

Чимитов Намжил

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.

Задачи

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 продемонстрировать их функционал.

Ход работы

Задача 1. Создан класс *Employee* с общими атрибутами и методом *get_info()*.

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

    def get_info(self):
        return f'Сотрудник {self.name} с идентификатором {self.id}.'
```

Задача 2. Создан класс *Manager* с атрибутом *department* и методом *manage_project()*.

```
class Manager(Employee):
    def __init__(self, name, id, department, **kwargs):
        super().__init__(name=name, id=id, **kwargs)
        self.department = department

    def manage_project(self):
        return f'Менеджер {self.name} работает над проектом в
департаменте {self.department}.'

    def get_info(self):
        base_info = super().get_info()
        return f'{base_info} Он менеджер в департаменте
{self.department}.'
```

Задача 3. Создан класс *Technician* с атрибутом *specialization* и методом *perform_maintenance()*.

```
class Technician(Employee):
    def __init__(self, name, id, specialization, **kwargs):
        super().__init__(name=name, id=id, **kwargs)
        self.specialization = specialization

    def perform_maintenance(self):
        return f'Техник {self.name} проводит техническое обслуживание
по специализации {self.specialization}.'

    def get_info(self):
        base_info = super().get_info()
        return f'{base_info} Техническая специализация:
{self.specialization}.'
```

Задача 4. Создан класс *TechManager*, который наследует как *Manager*, так и *Technician*.

```
class TechManager(Manager, Technician):
    def __init__(self, name, id, department, specialization,
**kwargs):
        super().__init__(name=name, id=id, department=department,
specialization=specialization, **kwargs)
        self.team = set()

    def get_info(self):
        base_info = super().get_info()
        return f'{base_info} Это технический менеджер.'
```

Задача 5. Добавлен метод *add_employee()* в *TechManager*.

```
class TechManager(Manager, Technician):
    def __init__(self, name, id, department, specialization,
**kwargs):
        super().__init__(name=name, id=id, department=department,
specialization=specialization, **kwargs)
        self.team = set()

    def add_employee(self, employee):
        self.team.add(employee)
        return f'Сотрудник {employee.name} добавлен в команду
{self.name}.'

    def get_info(self):
        base_info = super().get_info()
        return f'{base_info} Это технический менеджер.'
```

Задача 6. Реализован метод *get_team_info()* в *TechManager*.

```
class TechManager(Manager, Technician):
    def __init__(self, name, id, department, specialization,
**kwargs):
        super().__init__(name=name, id=id, department=department,
specialization=specialization, **kwargs)
        self.team = set()

    def add_employee(self, employee):
        self.team.add(employee)
        return f'Сотрудник {employee.name} добавлен в команду
{self.name}.'
```

```
    def get_team_info(self):
        team = f'Команда сотрудника {self.name}:\n'
        for employee in self.team:
            team += f'- {employee.get_info()}\n'
        return team

    def get_info(self):
        base_info = super().get_info()
        return f'{base_info} Это технический менеджер.'
```

Задача 7. Созданы объекты каждого класса и продемонстрирован их функционал.

```
employee = Employee('Антон', 1)
manager = Manager('Андрей', 2, 'HR')
technician = Technician('Алексей', 3, 'Электрик')
tech_manager = TechManager('Александр', 4, 'ИТ', 'Системное
администрирование')

print(employee.get_info())
print()

print(manager.get_info())
print(manager.manage_project())
print()

print(technician.get_info())
print(technician.perform_maintenance())
print()

print(tech_manager.get_info())
print(tech_manager.manage_project())
print(tech_manager.perform_maintenance())
print()

tech_manager.add_employee(employee)
tech_manager.add_employee(manager)
tech_manager.add_employee(technician)
print(tech_manager.get_team_info())
```

Вывод

В результате выполнения лабораторной работы разработана система управления сотрудниками, демонстрирующая множественное наследование, инкапсуляцию и полиморфизм в Python.