МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени
Федеральное государственное бюджетное общеобразовательное учреждение высшего образования

“Московский технический университет связи и информатики”
(МТУСИ)
Кафедра “Программная инженерия”

Лабораторная работа №7

по дисциплине “Введение в информационные технологии”

Выполнил: Студент группы
БПИ2503
Яричевский Даниил

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.

Задачи

1. Разработать систему управления сотрудниками

Ход работы

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

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

    def get_info(self):
        return f"Name: {self.name}, ID: {self.id}"

```

Рис. (1)

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

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

    def manage_project(self):
        return f"Сотрудник: {self.name}\nУправляет отделом: {self.department}"
```

Рис. (2)

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

```
17  class Techican(Employee):
18      def __init__(self, name, id, specialization):
19          super().__init__(name, id)
20          self.spec = specialization
21
22      def perfom_maintenance(self):
23          return f"Сотрудник: {self.name}\nВыполнил т.о. по: {self.spec}"
```

Рис. (3)

4 Создаем класс TechManager, который наследует как Manager, так и Technician.

```
34  class TechManager(Manager, Techican):
35      def __init__(self, name, id, department, specialization):
36          Employee.__init__(self, name, id)
37          self.department = department
38          self.spec = specialization
```

Рис. (4)

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

```
34  class TechManager(Manager, Techican):
35      def __init__(self, name, id, department, specialization):
36          Employee.__init__(self, name, id)
37          self.department = department
38          self.spec = specialization
39          self.employees = []
40
41      def add_employee(self, employee):
42          self.employees.append(employee)
43          return f'Сотрудник был привязан к {self.name}\n'
```

Рис. (5)

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

```
45      def get_team_info(self):
46          return f"В команде {self.name} следующие сотрудники: {self.employees}\n"
```

Рис. (6)

7 Создаем объекты каждого класса и демонстрируйте их функциональность.

```
print(e.get_info())
print(m.manage_project())
print(t.perfom_maintenance())
print(tm.add_employee(m))
print(tm.add_employee(e))
print(tm.get_team_info())
```

Рис. (7)

Вывод

Я разработал систему управления сотрудниками, демонстрирующую множественное наследование, инкапсуляцию и полиморфизм в Python.