Отчёт по лабораторной работе

Тема: Организация рабочего окружения и работа с Git. Стратегии ветвления и автоматизация контроля качества.

Сведения о студенте

Дата: 2025-12-20 **Семестр:** 3 **Группа:** ПИН-б-о-24-1 **Дисциплина:** Технологии программирования
Студент: Макаров Роман Дмитриевич

Оглавление

Введение

Цель работы

Разработка комплексной системы учета сотрудников компании с применением принципов объектно-ориентированного программирования, паттернов проектирования.

Используемые технологии

- **Язык программирования:** Python 3.x
- **Система контроля версий:** Git

Структура проекта

```
employee_management_system/
├── src/                          # Исходный код системы
│   ├── core/                     # Основные классы системы
│   │   ├── __init__.py
│   │   ├── abstract_employee.py  # Абстрактный класс AbstractEmployee
│   │   ├── employee.py           # Базовый класс Employee
│   │   ├── department.py         # Класс Department
```

```
│   │   ├── company.py            # Класс Company
│   │   └── project.py            # Класс Project
│   │
│   ├── employees/                # Классы сотрудников
│   │   ├── __init__.py
│   │   ├── manager.py            # Класс Manager
│   │   ├── developer.py          # Класс Developer
│   │   └── salesperson.py        # Класс Salesperson
│   │
│   ├── factories/                # Фабрики и порождающие паттерны
│   │   ├── __init__.py
│   │   ├── employee_factory.py   # EmployeeFactory
│   │   └── company_factory.py    # AbstractFactory для компаний
│   │
│   ├── patterns/                 # Реализации паттернов проектирования
│   │   ├── __init__.py
│   │   ├── singleton.py          # Singleton для DatabaseConnection
│   │   └── builder.py            # EmployeeBuilder
│   │
│   ├── utils/                    # Вспомогательные модули
│   │   ├── __init__.py
│   │   ├── comparators.py        # Компараторы
│   │   └── exceptions.py         # Кастомные исключения
│   │
│   └── database/                 # Работа с базой данных
│       ├── __init__.py
│       └── connection.py         # Singleton для подключения к БД
│
├── data/                         # Данные для тестирования
│   ├── json/                     # JSON файлы для сериализации
│   └── csv/                      # CSV отчеты
│
├── examples/                     # Примеры использования
│   └── demo_part5.py             # Демо Part 5: Паттерны
│
├── README.md                    # Описание проекта
└── main.py                      # Основной скрипт для запуска
```

Лабораторная работа 5: Применение паттернов проектирования

Цель

Рефакторинг системы с применением паттернов проектирования.

Реализованные паттерны

- **Singleton:** `DatabaseConnection`

- **Factory Method:** `EmployeeFactory`
- **Abstract Factory**: `CompanyFactory`
- **Builder:** `EmployeeBuilder`

Пример использования

```
# Демонстрация Singleton
    db1 = DatabaseConnection("test.db")
    db2 = DatabaseConnection("test.db")

    print(db1 is db2) # True

    # Получаем подключение
    conn1 = db1.get_connection() # Создано подключение к: test.db



        # Демонстрация фабрики сотрудников
    # Создаем менеджера
    man = ManagerFactory.create_employee(
        id=11,
        name="Фабричный Менеджер",
        department="SAL",
        base_salary=45000.0,
        bonus = 5000
    )
    # Создаем разработчика
    dev = DeveloperFactory.create_employee(
        id=12,
        name="Фабричный Разработчик",
        department="DEV",
        base_salary=60000.0,
        tech_stack=['JS', 'GO'],
        seniority_level='middle'
    )
    # Создаем продавца
    sal = SalespersonFactory.create_employee(
        id=13,
        name="Фабричный Продавец",
        department="SAL",
        base_salary=30000.0,
        commission_rate=0.2,
        sales_volume=200000.0
    )

        # Демонстрация фабрики компаний
    # Создаем техническую компанию
    tech_factory = TechCompanyFactory()
    tech_company = tech_factory.create_company("TechCompany")
    print(f"Создана IT-компания: {tech_company.name}  ({tech_company.__class__.__name__})")
    print(f"Отделы: {[d.name for d in tech_company.get_departments()]}")
    print(f"Проекты: {[p.name for p in tech_company.get_projects()]}\n")
```

```
# Создаем торговую компанию
sal_factory = SalesCompanyFactory()
sal_company = sal_factory.create_company("SalCompany")
print(f"Создана торговая компания: {sal_company.name}  ({sal_company.__class__.__name__})")
print(f"Отделы: {[d.name for d in sal_company.get_departments()]}")
print(f"Проекты: {[p.name for p in sal_company.get_projects()]}")



    # Демонстрация Builder
# Создаем разработчика через Builder
developer = (EmployeeBuilder()
             .set_id(1)
             .set_name("Bob Smith")
             .set_department("DEV")
             .set_base_salary(5000)
             .set_type("developer")
             .set_tech_stack(["Python", "SQL"])
             .set_seniority_level("senior")
             .build())

# Создаем менеджера через Builder
manager = (EmployeeBuilder()
           .set_id(2)
           .set_name("Alice Johnson")
           .set_department("MANAGEMENT")
           .set_base_salary(7000)
           .set_type("manager")
           .set_bonus(2000)
           .build())
```

Заключение

Достигнутые результаты

1. Разработана полнофункциональная система учета сотрудников
2. Применены все принципы ООП: инкапсуляция, наследование, полиморфизм
3. Реализованы 4 паттерна проектирования
4. Создана расширяемая и поддерживаемая архитектура

Преимущества реализованного решения

- **Гибкость:** Легкое добавление новых типов сотрудников
- **Масштабируемость:** Поддержка большого количества сотрудников и отделов
- **Поддерживаемость:** Чистая архитектура и документация

Возможности дальнейшего развития

- Интеграция с веб-интерфейсом

- Добавление модуля отчетности
- Поддержка распределенной архитектуры
- Интеграция с системами аутентификации

Приложения

Приложение А: Примеры использования

`employee_management_system/exapmles/`:

- demo_patterns.py - Демонстрация работы паттернов

Список использованных источников

1. Роберт Мартин. "Чистый код. Создание, анализ и рефакторинг"
2. Мартин Фаулер. "Рефакторинг. Улучшение существующего кода"
3. Эрик Гамма и др. "Паттерны объектно-ориентированного проектирования"
4. Документация Python: https://docs.python.org/3/
5. Документация pytest: https://docs.pytest.org/