Отчёт по лабораторной работе

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

Сведения о студенте

Дата: 2025-12-24 **Семестр:** 3 семестр **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 и современных практик тестирования.

Используемые технологии

- **Язык программирования:** Python 3.x
- **Фреймворки:** pytest, unittest
- **Инструменты:** pylint, black, mypy
- **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
│
├── tests/                        # Тесты для всех частей ЛР
│   ├── __init__.py
│   ├── conftest.py               # Фикстуры pytest
│   │
│   ├── test_core/                # Тесты основных классов
│   │   ├── __init__.py
│   │   ├── test_employee.py      # Тесты Part 1: Инкапсуляция
│   │   ├── test_department.py    # Тесты Part 3: Полиморфизм
│   │   └── test_company.py       # Тесты Part 4: Композиция
│   │
│   ├── test_employees/           # Тесты классов сотрудников
│   │   ├── __init__.py
│   │   ├── test_employees_hierarchy.py  # Тесты Part 2: Наследование
│   │   ├── test_manager.py
│   │   ├── test_developer.py
│   │   └── test_salesperson.py
│   │
│   └── test_patterns/            # Тесты паттернов
│       ├── __init__.py
│       ├── test_singleton.py
│       ├── test_factory.py
```

```
│       └── test_builder.py
│
├── report/                        # Отчеты от линтеров и тестов
│   ├── black.report
│   ├── mypy.report
│   ├── pylint.report
│   └── test.py
│
├── README.md                      # Описание проекта
└── main.py                        # Основной скрипт для запуска
```

Лабораторная работа 9: Принципы поддержания качества кода

Цель

Улучшение качества кода через рефакторинг и применение принципов SOLID.

Проведенный рефакторинг

Применение SOLID

- **SRP:** Разделение ответственности классов
- **ISP:** Разделение интерфейсов

Разделение ответственности классов (SRP)

- **SRP:** Каждый валидатор отвечает только за валидацию конкретной сущности. Устраняет дублирование валидационного кода.

Код до применения SRP:

- Валидаторы разбросаны по разным файлам, имеется значительное дублирование кода

```
class AbstractEmployee(ABC):
    def __init__(self, id: int, name: str, department: str, base_salary: float):

        self.__id = id
        self.__name = name
        self.__department = department
        self.__base_salary = base_salary

        self._validate_id(id)
        self._validate_name(name)
        self._validate_department(department)
        self._validate_base_salary(base_salary)

    def _validate_id(self, value: int) -> None:
```

```
        """Валидация ID."""
        if not isinstance(value, int) or value <= 0:
            raise ValueError(f"ID должен быть целым положительным числом!")

    def _validate_name(self, value: str) -> None:
        """Валидация имени."""
        if not isinstance(value, str):
            raise ValueError("Имя должно быть строкой!")
        if not value.strip():
            raise ValueError("Имя не может быть пустой строкой!")
```

Код после применения SRP:

- Код с проверками собран в одном файле, структурирован, помогает значительно избежать повторений при использовании валидаторов

```
class BaseValidator:

    @staticmethod
    def validate_not_empty_string(value: Any, field_name: str) -> str:
        if not isinstance(value, str):
            raise ValueError(f"{field_name} должно быть строкой!")
        if not value.strip():
            raise ValueError(f"{field_name} не может быть пустой строкой!")
        return value.strip()

    @staticmethod
    def validate_positive_number(value: Any, field_name: str) -> float:
        if not isinstance(value, (int, float)):
            raise ValueError(f"{field_name} должно быть числом!")
        if value <= 0:
            raise ValueError(f"{field_name} должно быть положительным!")
        return float(value)
```

2.4. Принцип разделения интерфейса (ISP):

- **ISP:** Разделение интерфейсов

Код до применения ISP:

- Разделение абстрактных методов отстутствует, код строго связан, "большой" интерфейс класса

```
    class AbstractEmployee(ABC):
    def __init__(self, id: int, name: str, department: str, base_salary: float):
        self.__id = id
        self.__name = name
        # ...

    @classmethod
    @abstractmethod
```

```
    def from_dict(cls, data: dict) -> "AbstractEmployee":
        pass

    @abstractmethod
    def calculate_salary(self) -> float:
        pass

    @abstractmethod
    def get_info(self) -> str:
        pass

    @abstractmethod
    def to_dict(self) -> dict:
        pass
```

Код после применения ISP:

- Создан узкий, специфичный интерфейс, с раздельными методами, что повышает гибкость и снижает сложность поддержки системы.

```
class ISalaryCalculable(ABC):
    @abstractmethod
    def calculate_salary(self) -> float:
        pass

class IInfoProvidable(ABC):
    @abstractmethod
    def get_info(self) -> str:
        pass

class IToDict(ABC):
    @abstractmethod
    def to_dict(self) -> dict:
        pass

class IFromDict(ABC):
    @classmethod
    @abstractmethod
    def from_dict(cls, data: dict) -> "AbstractEmployee":
        pass
```

Улучшение метрик

Метрика	До рефакторинга	После рефакторинга
Cyclomatic complexity	2.02	1.8
Pylint score	7.59/10	9.1/10

Инструменты качества

- Настроены линтеры: `pylint`, `black`, `mypy`

- Внедрены pre-commit хуки
- Настроен CI/CD pipeline

Заключение

Достигнутые результаты

1. Разработана полнофункциональная система учета сотрудников
2. Применены все принципы ООП: инкапсуляция, наследование, полиморфизм
3. Реализованы 4+ паттернов проектирования
4. Обеспечено высокое качество кода через тестирование и рефакторинг
5. Создана расширяемая и поддерживаемая архитектура

Преимущества реализованного решения

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

Возможности дальнейшего развития

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

Приложения

Приложение A: Результаты тестирования

Подробные отчеты pytest

Приложение B: Метрики качества кода

Отчеты pylint, black и mypy

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/