Отчет по лабораторной работе 9

Рефакторинг системы учета сотрудников

Дата: 30-12-2025
Семестр: 2 курс, 1 семестр
Группа: ПИН-Б-О-24-2
Дисциплина: Технологии программирования
Студент: Осипов Александр Сергеевич

Цель работы

Освоить практические навыки рефакторинга существующего кода, применения принципов SOLID, устранения "запахов кода" и улучшения архитектуры системы.

Теоретическая часть

Принципы SOLID:

- **SRP (Single Responsibility)** - класс должен иметь только одну причину для изменения
- **OCP (Open/Closed)** - открыт для расширения, закрыт для изменения
- **LSP (Liskov Substitution)** - подклассы должны заменять базовые классы
- **ISP (Interface Segregation)** - много специализированных интерфейсов лучше одного общего
- **DIP (Dependency Inversion)** - зависимость от абстракций, а не от конкретных классов

Практическая часть

Часть 1: Анализ кода

Результат pylint до рефакторинга:

```
employee_system.py:9:0: R0902: Too many instance attributes (11/7)
employee_system.py:12:4: R0913: Too many arguments (11/5)
employee_system.py:12:23: W0622: Redefining built-in 'id'
employee_system.py:12:58: W0622: Redefining built-in 'type'
employee_system.py:27:4: R0911: Too many return statements (8/6)
-----------------------------------
Your code has been rated at 9.06/10
```

Выявленные "запахи кода":

1. **Длинный метод** - `calculate_salary()` с множеством условий if-elif
2. **Большой класс** - `Employee` имеет 11 атрибутов и множество обязанностей
3. **Нарушение SRP** - валидация, сериализация, логирование в одном классе
4. **Нарушение OCP** - для добавления типа сотрудника нужно менять код метода
5. **Дублирование кода** - повторяющаяся логика в `Department` и `Company`

Часть 2: Применение принципов SOLID

2.1. SRP - Единственная ответственность

Было: Все в одном классе Employee

```
class Employee:
    def calculate_salary(self): ...
    def validate(self): ...
    def save_to_file(self): ...
    def log_action(self): ...
```

Стало: Отдельные классы

```
class Employee:           # только данные сотрудника
class EmployeeValidator:  # валидация
```

```
class JsonSerializer:     # сериализация
class ReportGenerator:    # отчеты
```

2.2. OCP - Открытость/закрытость

Было: Длинный if-elif для расчета зарплаты

```
def calculate_salary(self):
    if self.type == "manager":
        return self.base_salary + self.bonus
    elif self.type == "developer":
        if self.level == "junior":
            return self.base_salary * 1.0
        # ... много условий
```

Стало: Паттерн Strategy

```
class SalaryStrategy(ABC):
    @abstractmethod
    def calculate(self, base_salary, **kwargs) -> float: pass

class DeveloperSalaryStrategy(SalaryStrategy):
    def calculate(self, base_salary, **kwargs) -> float:
        level = kwargs.get('level', 'junior')
        return base_salary * self.LEVEL_COEFFICIENTS[level]
```

2.3. ISP - Разделение интерфейсов

Создано:

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

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

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

2.4. DIP - Инверсия зависимостей

Создано:

```
class IEmployeeRepository(ABC):
    @abstractmethod
    def save(self, employee): pass
    @abstractmethod
    def find_by_id(self, emp_id): pass

class InMemoryEmployeeRepository(IEmployeeRepository):
    # Реализация

class FileEmployeeRepository(IEmployeeRepository):
    # Реализация
```

Часть 3: Структура после рефакторинга

```
after_refactoring/
├── interfaces.py        # ISP - интерфейсы
├── salary_strategies.py # OCP - стратегии расчета ЗП
├── employees.py         # Классы сотрудников (LSP)
├── validators.py        # SRP - валидация
├── serializers.py       # SRP - сериализация
├── repositories.py      # DIP - репозитории
├── department.py        # Класс отдела
├── managers.py          # SRP - DepartmentManager, FinancialCalculator,
ReportGenerator
└── main.py              # Демонстрация
```

Результаты выполнения

Пример работы программы (после рефакторинга)

```
=== Информация о сотрудниках ===
Разработчик: Иван, Уровень: senior, Стек: [Python, JS], ЗП: 100000.0
Разработчик: Мария, Уровень: middle, Стек: [Java], ЗП: 67500.0
Менеджер: Петр, Бонус: 20000, ЗП: 100000
Продавец: Анна, Комиссия: 10.0%, Продажи: 100000, ЗП: 50000.0
Сотрудник: Сергей, Отдел: Продажи, ЗП: 35000
```

```
=== Финансовые расчеты ===
Общий ФОТ: 352500.0
Средняя ЗП: 70500.00

=== Отчет ===
================================================
ОТЧЕТ ПО КОМПАНИИ
================================================
ОТДЕЛЫ:
------------------------------
  IT: 3 сотр., ФОТ: 267500.0
  Продажи: 2 сотр., ФОТ: 85000.0
ИТОГО:
------------------------------
  Всего сотрудников: 5
  Общий ФОТ: 352500.0
```

Сравнение метрик

Метрика	До	После
Файлов	1	9
Классов	3	15+
Строк кода	~250	~350
Макс. атрибутов класса	11	5
Макс. аргументов метода	11	7
Принципов SOLID	0	5

Выводы

1. Применение SRP позволило разделить ответственности между классами - теперь каждый класс имеет одну причину для изменения
2. Использование паттерна Strategy (OCP) позволяет добавлять новые типы сотрудников без изменения существующего кода

3. Разделение интерфейсов (ISP) делает классы более гибкими - можно реализовать только нужные методы
4. Инверсия зависимостей (DIP) позволяет легко менять реализации репозиториев
5. Код стал более читаемым и поддерживаемым, хотя объем увеличился

Ответы на контрольные вопросы

1. **Что такое рефакторинг?** - Изменение структуры кода без изменения его поведения для улучшения читаемости и поддерживаемости
2. **Какие "запахи кода" вы знаете?** - Длинный метод, большой класс, дублирование кода, длинный список параметров
3. **Что означает принцип SRP?** - Класс должен иметь только одну причину для изменения, т.е. одну ответственность
4. **Как паттерн Strategy помогает соблюдать OCP?** - Позволяет добавлять новые алгоритмы (стратегии) без изменения классов, которые их используют

Приложения

- Исходный код до рефакторинга: `project/before_refactoring/`
- Исходный код после рефакторинга: `project/after_refactoring/`