Отчет по лабораторной работе 8

Тестирование программного обеспечения

Дата: 28-12-2025
Семестр: 2 курс, 1 семестр
Группа: ПИН-Б-О-24-2
Дисциплина: Технологии программирования
Студент: Осипов Александр Сергеевич

Цель работы

Освоить основы модульного тестирования, научиться писать unit-тесты для классов и методов, использовать фреймворк pytest, применять техники изоляции зависимостей с помощью моков.

Теоретическая часть

Unit-тестирование - метод тестирования, при котором проверяются отдельные модули программы. Использован фреймворк pytest и библиотека unittest.mock.

Основные концепции: - Arrange-Act-Assert (AAA) - структура теста - Параметризация тестов - Mock-объекты для изоляции зависимостей - Фикстуры для подготовки тестового окружения

Практическая часть

Выполненные задачи

- [x] Часть 1: Тестирование класса Employee (инкапсуляция, валидация)
- [x] Часть 2: Тестирование иерархии классов (наследование, полиморфизм)

- [x] Часть 3: Тестирование магических методов и Department
- [x] Часть 4: Тестирование Project и Company (композиция, агрегация)
- [x] Часть 5: Тестирование паттернов проектирования

Ключевые фрагменты кода

Тест валидации Employee:

```
def test_employee_invalid_id_raises_error(self):
    with pytest.raises(ValueError):
        Employee(-1, "Alice", "IT", 5000)

def test_employee_empty_name_raises_error(self):
    with pytest.raises(ValueError):
        Employee(1, "", "IT", 5000)
```

Параметризованный тест Developer:

```
@pytest.mark.parametrize("level,expected_salary", [
    ("junior", 5000),
    ("middle", 7500),
    ("senior", 10000)
])
def test_developer_salary_by_level(self, level, expected_salary):
    dev = Developer(1, "Alice", "DEV", 5000, ["Python"], level)
    assert dev.calculate_salary() == expected_salary
```

Тест паттерна Singleton:

```
def test_singleton_same_instance(self):
    db1 = DataStorage.get_instance()
    db2 = DataStorage.get_instance()
    assert db1 is db2
    assert id(db1) == id(db2)
```

Тест паттерна Observer с Mock:

```
def test_observer_with_mock(self):
    emp = Employee(1, "John", "IT", 5000)
    mock_observer = Mock()
    emp.add_observer(mock_observer)
```

```
    emp.base_salary = 6000
    mock_observer.update.assert_called_once()
```

Результаты выполнения

Запуск тестов

```
 $ pytest tests/ -v
============================= test session starts ==============================
collected 112 items

tests/test_department.py ......................                        [ 20%]
tests/test_employee.py .....................                           [ 40%]
tests/test_employees_hierarchy.py .......................              [ 60%]
tests/test_patterns.py .........................                       [ 83%]
tests/test_project_company.py ..................                       [100%]

============================== 112 passed in 0.64s ==============================
```

Тестирование

- [x] Все 112 тестов пройдены
- [x] Тесты валидации работают корректно
- [x] Параметризованные тесты покрывают разные сценарии
- [x] Mock-объекты успешно изолируют зависимости

Выводы

1. Написаны unit-тесты для всех частей системы учета сотрудников
2. Использованы параметризованные тесты для проверки различных сценариев
3. Применены mock-объекты для тестирования паттерна Observer
4. Все 112 тестов успешно проходят

Ответы на контрольные вопросы

1. **Что такое unit-тест?** - Тест, проверяющий работу отдельного модуля (класса, функции) изолированно от других частей системы.
2. **Зачем нужна параметризация тестов?** - Позволяет запускать один и тот же тест с разными входными данными, уменьшая дублирование кода.
3. **Что такое mock-объект?** - Имитация реального объекта, позволяющая изолировать тестируемый код от внешних зависимостей.

Приложения

- Исходный код тестов
- Исходный код моделей
- Исходный код паттернов