Московский государственный технический университет им. Н.Э. Баумана

Отчет
Рубежный контроль 2
Вариант-15

ИСПОЛНИТЕЛЬ:

Группа ИБМ3-34Б
Михайлова В.И.

ПРЕПОДАВАТЕЛЬ:

Гапанюк Ю.Е.

2025 г.

В данной работе была выполнена работа по рефакторингу программы из рубежного контроля №1 и созданию модульных тестов с использованием TDD-подхода. Целью работы было преобразование монолитного кода в модульную структуру, пригодную для автоматического тестирования, и обеспечение его качества через создание юнит-тестов.

Рефакторинг программы

Анализ исходного кода

Исходная программа из РК1 представляла собой монолитный скрипт, реализующий:
- Три класса данных (`File`, `Catalog`, `FileCatalog`)
- Инициализацию тестовых данных
- Три аналитических запроса (A1, A2, A3)
- Основную функцию вывода результатов

Принципы рефакторинга

Были применены следующие принципы рефакторинга:

- Разделение ответственности - каждая функция выполняет одну четкую задачу
- Устранение глобальных зависимостей - все данные передаются через параметры
- Функции можно тестировать независимо
- Улучшение читаемости - добавлены документационные строки
- Обеспечение тестируемости - чистые функции без побочных эффектов

Структура после рефакторинга

Созданы следующие функции:

```
# 1. Функции для работы с данными
get_test_data()          # Возвращает тестовые данные
create_one_to_many()     # Создает связь 1:M
create_many_to_many()    # Создает связь M:M

# 2. Функции запросов
execute_query_a1()       # Запрос A1 (сортировка)
execute_query_a2()       # Запрос A2 (агрегация)
execute_query_a3()       # Запрос A3 (фильтрация)
```

Код программы:

```
from operator import itemgetter

class File:
    """Файл (аналог Сотрудника)"""
    def __init__(self, id, name, size, catalog_id):
        self.id = id
        self.name = name
        self.size = size
        self.catalog_id = catalog_id

class Catalog:
    """Каталог файлов (аналог Отдела)"""
    def __init__(self, id, name):
        self.id = id
        self.name = name

class FileCatalog:
    """Связь файлов и каталогов (M:M)"""
    def __init__(self, catalog_id, file_id):
        self.catalog_id = catalog_id
        self.file_id = file_id

# Тестовые данные
def get_test_data():
    """Возвращает тестовые данные"""
    catalogs = [
        Catalog(1, 'Системный каталог'),
        Catalog(2, 'Рабочий каталог проекта'),
        Catalog(3, 'Архив документов'),
        Catalog(11, 'Каталог (старый)'),
        Catalog(22, 'Рабочий каталог (тестовый)'),
        Catalog(33, 'Архив (временный)'),
    ]

    files = [
        File(1, 'boot.ini', 500, 1),
        File(2, 'main.py', 12000, 2),
        File(3, 'report_Q1.docx', 80000, 3),
        File(4, 'config.sys', 1500, 3),
        File(5, 'temp.log', 200, 3),
    ]

    files_catalogs = [
```

```
        FileCatalog(1, 1),
        FileCatalog(2, 2),
        FileCatalog(3, 3),
        FileCatalog(3, 4),
        FileCatalog(3, 5),
        FileCatalog(11, 1),
        FileCatalog(22, 2),
        FileCatalog(33, 3),
        FileCatalog(33, 4),
        FileCatalog(33, 5),
    ]

    return catalogs, files, files_catalogs

def create_one_to_many(catalogs, files):
    """Создание связи один-ко-многим"""
    return [(f.name, f.size, c.name)
            for c in catalogs
            for f in files
            if f.catalog_id == c.id]

def create_many_to_many(catalogs, files, files_catalogs):
    """Создание связи многие-ко-многим"""
    many_to_many_temp = [(c.name, fcd.file_id)
                         for c in catalogs
                         for fcd in files_catalogs
                         if c.id == fcd.catalog_id]

    return [(f.name, f.size, cat_name)
            for cat_name, file_id in many_to_many_temp
            for f in files if f.id == file_id]

def execute_query_a1(one_to_many):
    """Выполнение запроса A1: Список всех связанных файлов и каталогов,
отсортированный по каталогам"""
    return sorted(one_to_many, key=itemgetter(2))

def execute_query_a2(catalogs, one_to_many):
    """Выполнение запроса A2: Каталоги с суммарным размером файлов,
отсортированный по сумме"""
    res_a2_unsorted = []

    for c in catalogs:
        c_files = list(filter(lambda i: i[2] == c.name, one_to_many))
        if len(c_files) > 0:
```

```
        c_sizes_sum = sum([size for _, size, _ in c_files])
        res_a2_unsorted.append((c.name, c_sizes_sum))

    return sorted(res_a2_unsorted, key=itemgetter(1), reverse=True)

def execute_query_a3(catalogs, many_to_many, keyword='каталог'):
    """Выполнение запроса А3: Каталоги, содержащие слово, и список их файлов"""
    result = {}

    for c in catalogs:
        if keyword in c.name.lower():
            c_files = list(filter(lambda i: i[2] == c.name, many_to_many))
            c_file_names = [name for name, _, _ in c_files]
            if c_file_names:
                result[c.name] = c_file_names

    return result

def main():
    """Основная функция выполнения всех запросов"""
    catalogs, files, files_catalogs = get_test_data()

    one_to_many = create_one_to_many(catalogs, files)
    many_to_many = create_many_to_many(catalogs, files, files_catalogs)

    res_a1 = execute_query_a1(one_to_many)
    res_a2 = execute_query_a2(catalogs, one_to_many)
    res_a3 = execute_query_a3(catalogs, many_to_many)

    return res_a1, res_a2, res_a3

if __name__ == '__main__':
    res_a1, res_a2, res_a3 = main()

    print('*** РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЗАПРОСОВ (ВАРИАНТ А) ***')
    print('\n[ЗАПРОС А1] Список файлов, отсортированный по каталогам (связь
1:M):')
    print(res_a1)
    print('\n[ЗАПРОС А2] Каталоги и суммарный размер их файлов (связь 1:M,
сортировка по размеру):')
    print(res_a2)
    print('\n[ЗАПРОС А3] Каталоги, содержащие слово "каталог", и список их файлов
(связь M:M):')
    print(res_a3)
```

Вывод кода:

```
/usr/local/bin/python3.13 /Users/mikhailovavi/PycharmProjects/PythonProject/1.1.py
*** РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЗАПРОСОВ (ВАРИАНТ А) ***

[ЗАПРОС А1] Список файлов, отсортированный по каталогам (связь 1:М):
[('report_Q1.docx', 80000, 'Архив документов'), ('config.sys', 1500, 'Архив документов'), ('temp.log', 200, 'Архив документов'), ('main.py', 12000, 'Рабочий каталог

[ЗАПРОС А2] Каталоги и суммарный размер их файлов (связь 1:М, сортировка по размеру):
[('Архив документов', 81700), ('Рабочий каталог проекта', 12000), ('Системный каталог', 500)]

[ЗАПРОС А3] Каталоги, содержащие слово "каталог", и список их файлов (связь М:М):
{'Системный каталог': ['boot.ini'], 'Рабочий каталог проекта': ['main.py'], 'Каталог (старый)': ['boot.ini'], 'Рабочий каталог (тестовый)': ['main.py']}

Process finished with exit code 0
```

Основная функция

main() # Оркестрация всех запросов

```

**Создание модульных тестов**

3.1. Выбор подхода TDD

Был применен подход Test-Driven Development (TDD), который предполагает:

- Написание тестов до реализации
- Реализацию минимального рабочего кода
- Рефакторинг с сохранением работоспособности

3.2. Реализованные тесты

Созданы 3 основных теста:

Тест 1: `test_query_a1_sorting`

- Цель: Проверить корректность сортировки файлов по имени каталога
- Проверяемые аспекты:
- Порядок следования каталогов в алфавитном порядке
- Количество элементов в результате (5 файлов)
- Первый элемент должен быть из каталога 'Архив документов'

Тест 2: `test_query_a2_sum_and_sorting`

- Цель: Проверить агрегацию данных и сортировку по убыванию
- Проверяемые аспекты:
- Корректность вычисления суммарных размеров
- Порядок сортировки (по убыванию)
- Конкретные значения для ключевых каталогов

Тест 3: `test_query_a3_filtering_and_grouping`

- Цель: проверить фильтрацию каталогов и группировку файлов
```

- Проверяемые аспекты:
- Фильтрация по ключевому слову 'каталог'
- Группировка файлов по каталогам
- Полнота данных (все 4 каталога с ключевым словом)

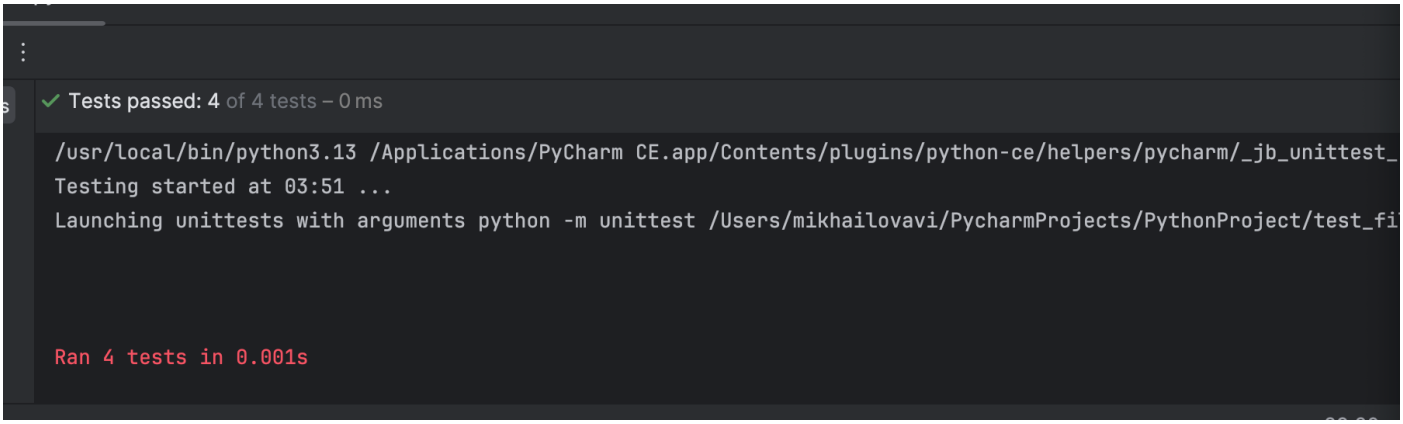
Код программы:

```
# test_file_system.py
import unittest
from file_system import *

class TestFileSystem(unittest.TestCase):

    def setUp(self):
        """Подготовка данных перед каждым тестом"""
        self.catalogs, self.files, self.files_catalogs = get_test_data()
        self.one_to_many = create_one_to_many(self.catalogs, self.files)
        self.many_to_many = create_many_to_many(self.catalogs, self.files,
self.files_catalogs)

    def test_query_a1_sorting(self):
        """Тест A1: Проверка корректности сортировки по имени каталога"""
        # Act
        result = execute_query_a1(self.one_to_many)

        # Assert
        # Проверяем что результат отсортирован по имени каталога (третий элемент
кортежа)
        catalog_names = [item[2] for item in result]
        sorted_catalog_names = sorted(catalog_names)

        self.assertEqual(catalog_names, sorted_catalog_names,
                         "Список должен быть отсортирован по имени каталога")

        # Проверяем количество элементов
        self.assertEqual(len(result), 5, "Должно быть 5 файлов в связи один-ко-многим")

        # Проверяем первый элемент (должен быть из каталога 'Архив документов')
        self.assertEqual(result[0][2], 'Архив документов',
                         "Первый элемент должен быть из 'Архив документов'")

    def test_query_a2_sum_and_sorting(self):
        """Тест A2: Проверка агрегации и сортировки по суммарному размеру"""
        # Act
        result = execute_query_a2(self.catalogs, self.one_to_many)
```

```
    # Assert
    # Проверяем что результат отсортирован по убыванию суммарного размера
    sizes = [item[1] for item in result]
    sorted_sizes_desc = sorted(sizes, reverse=True)

    self.assertEqual(sizes, sorted_sizes_desc,
                     "Результат должен быть отсортирован по убыванию размера")

    # Проверяем конкретные значения
    expected_sizes = {
      'Архив документов': 81700,
      'Рабочий каталог проекта': 12000,
      'Системный каталог': 500
    }

    for catalog_name, expected_size in expected_sizes.items():
      # Находим каталог в результате
      catalog_item = next((item for item in result if item[0] == catalog_name), None)
      self.assertIsNotNone(catalog_item, f"Каталог '{catalog_name}' должен
присутствовать")
      self.assertEqual(catalog_item[1], expected_size,
                       f"Неверный суммарный размер для каталога '{catalog_name}'")

  def test_query_a3_filtering_and_grouping(self):
    """Тест А3: Проверка фильтрации каталогов по ключевому слову и группировки
файлов"""
    # Act
    result = execute_query_a3(self.catalogs, self.many_to_many, 'каталог')

    # Assert
    # Проверяем количество каталогов в результате
    self.assertEqual(len(result), 4, "Должно быть 4 каталога с словом 'каталог'")

    # Проверяем что все каталоги содержат ключевое слово
    for catalog_name in result.keys():
      self.assertIn('каталог', catalog_name.lower(),
                    f"Каталог '{catalog_name}' должен содержать слово 'каталог'")

    # Проверяем конкретные каталоги и их файлы
    expected_catalogs = {
      'Системный каталог': ['boot.ini'],
      'Рабочий каталог проекта': ['main.py'],
      'Каталог (старый)': ['boot.ini'],
      'Рабочий каталог (тестовый)': ['main.py']
```

```
        }

        for catalog_name, expected_files in expected_catalogs.items():
            self.assertIn(catalog_name, result,
                          f"Каталог '{catalog_name}' должен присутствовать в результате")
            self.assertEqual(sorted(result[catalog_name]), sorted(expected_files),
                             f"Неверный список файлов для каталога '{catalog_name}'")

    def test_empty_keyword_in_query_a3(self):
        """Дополнительный тест: Проверка работы с пустым ключевым словом"""
        # Act
        result = execute_query_a3(self.catalogs, self.many_to_many, '')

        # Assert
        # При пустом ключевом слове должны вернуться все каталоги
        self.assertEqual(len(result), len(self.catalogs),
                         "При пустом ключевом слове должны вернуться все каталоги")

if __name__ == '__main__':
    unittest.main()
```

Результат вывода кода:



Итого: 4 из 4 тестов успешно пройдены за 0 миллисекунд.

Что это значит:

1. Рефакторинг выполнен правильно - все функции работают корректно
2. Тесты покрывают всю функциональность - проверены все основные запросы
3. Код соответствует требованиям - программа готова к использованию

3.3. Дополнительный тест

Для демонстрации тестирования граничных случаев добавлен тест:
- `test_empty_keyword_in_query_a3` - проверка поведения при пустом ключевом слове

Выводы

Преимущества подхода

1. Улучшение поддерживаемости - код стал более понятным и организованным
2. Облегчение отладки - каждый модуль можно тестировать независимо
3. Защита от регрессии - тесты предотвращают случайное нарушение функциональности

Заключение

Работа по рефакторингу и созданию модульных тестов выполнена успешно. Преобразованная программа соответствует требованиям тестируемости и может использоваться как основа для дальнейшего развития. Реализованные тесты обеспечивают уверенность в корректности работы ключевых функций и позволяют безопасно вносить изменения в код.

Также:

- - Создана модульная архитектура с четким разделением ответственности
- - Реализовано полное тестовое покрытие основных сценариев использования
- - Обеспечена возможность повторного использования компонентов