Рубежный контроль №2 по дисциплине: «Парадигмы и конструкции языков программирования»

Рытикова Кирена, ИБМ3-34Б

Условия рубежного контроля №2 по курсу ПиК ЯП

Рубежный контроль представляет собой разработку тестов на языке Python.

1) Проведите рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования.

2) Для текста программы рубежного контроля №1 создайте модульные тесты с применением TDD - фреймворка (3 теста).

Текст программы:

```
from operator import itemgetter

class Detail:
    '''Деталь'''
    def __init__(self, id, name, price, prod_id):
        self.id = id
        self.name = name
        self.price = price
        self.prod_id = prod_id
    def __repr__(self):
        return f"Detail(id={self.id}, name='{self.name}', price={self.price},
prod_id={self.prod_id})"
class Producer:
    '''Производитель'''
    def __init__(self, id, name):
        self.id = id
        self.name = name
    def __repr__(self):
        return f"Producer(id={self.id}, name='{self.name}')"

class DetailProducer:
    '''связь многие ко многим'''
    def __init__(self, prod_id, detail_id):
        self.prod_id = prod_id
        self.detail_id = detail_id
    def __repr__(self):
        return f"DetailProducer(prod_id={self.prod_id},
detail_id={self.detail_id})"

'''функции для обработки данных для тестирования'''
def get_one_to_many(producers, details):
    return [(d.name, d.price, p.name)
            for p in producers
            for d in details
            if d.prod_id == p.id]
def get_many_to_many(producers, details, details_producers):
    many_to_many_temp = [(p.name, dp.prod_id, dp.detail_id)
                         for p in producers
                         for dp in details_producers
                         if p.id == dp.prod_id]
    return [(d.name, d.price, prod_name)
            for prod_name, prod_id, detail_id in many_to_many_temp
            for d in details if d.id == detail_id]

def task_a1(producers, details):
    '''Задание А1: Список всех связанных деталей и производителей,
отсортированный
```

```
    по производителям (связь один-ко-многим)'''
    one_to_many = get_one_to_many(producers, details)
    return sorted(one_to_many, key=itemgetter(2))

def task_a2(producers, details):
    '''Задание А2: Список производителей с суммарной стоимостью деталей,
    отсортированный по суммарной цене (связь один-ко-многим)'''
    one_to_many = get_one_to_many(producers, details)
    res_2_unsorted = []
    for p in producers:
        p_details = list(filter(lambda i: i[2] == p.name, one_to_many))
        if len(p_details) > 0:
            prices = [price for _, price, _ in p_details]
            total_price = sum(prices)
            res_2_unsorted.append((p.name, total_price))
    # сортировка по суммарной цене
    return sorted(res_2_unsorted, key=itemgetter(1), reverse=True)

def task_a3(producers, details, details_producers):
    '''Задание А3: Список производителей, у которых в названии есть слово
'завод',
    и их детали (связь многие-ко-многим).'''
    many_to_many = get_many_to_many(producers, details, details_producers)
    res_3 = {}
    for p in producers:
        if "завод" in p.name.lower():
            p_details = list(filter(lambda i: i[2] == p.name, many_to_many))
            p_detail_names = [x for x, _, _ in p_details]
            res_3[p.name] = p_detail_names
    return res_3

'''Производители'''
producers = [
    Producer(1, "Завод деталей"),
    Producer(2, "Машиностроительный завод"),
    Producer(3, "Фабрика Механическая"),
    Producer(4, "Компания ТехДеталь"),
    Producer(5, "Завод МеталлПром"),
    Producer(6, "Механический цех Деталь")
]

'''Детали'''
details = [
    Detail(1, "Болт", 10, 1),
    Detail(2, "Гайка", 8, 1),
    Detail(3, "Шайба", 7, 2),
    Detail(4, "Подшипник", 50, 3),
    Detail(5, "Поршень", 100, 3),
    Detail(6, "Втулка", 40, 4),
    Detail(7, "Шестерня", 120, 5),
    Detail(8, "Клапан", 90, 6),
    Detail(9, "Фильтр масляный", 110, 6),
]

''' связи многие ко многим '''
details producers = [
    DetailProducer(1, 1),
    DetailProducer(1, 2),
    DetailProducer(2, 3),
    DetailProducer(3, 4),
    DetailProducer(3, 5),
    DetailProducer(4, 6),
    DetailProducer(4, 1),
    DetailProducer(2, 2),
```

```
    DetailProducer(5, 7),
    DetailProducer(6, 8),
    DetailProducer(6, 9),
    DetailProducer(2, 4),
    DetailProducer(3, 6),
    DetailProducer(5, 1),
]

def print_results(producers, details, details_producers):
    '''Вывод результатов'''
    # Задание A1
    print('Задание A1')
    print("Список всех связанных деталей и производителей, отсортированный по производителям:")
    res_1 = task_a1(producers, details)
    print(res_1)
    # Задание A2
    print('\nЗадание A2')
    print("Список производителей с суммарной стоимостью деталей, отсортированный по суммарной цене:")
    res_2 = task_a2(producers, details)
    print(res_2)
    # Задание A3
    print('\nЗадание A3')
    print("Список производителей, у которых в названии есть слово 'завод', и их детали:")
    res_3 = task_a3(producers, details, details_producers)
    print(res_3)

if __name__ == "__main__":
    print_results(producers, details, details_producers)
```

Был выполнен рефакторинг исходного кода из РК1 (введена объектно-ориентированная структура данных, логика обработки внесена в отдельные функции, код стал структурированным и расширяемым).

А также проведены тесты:

```
import unittest
import meow2

class TestMeow2Functions(unittest.TestCase):
    def test_task_a1_sorted_by_producer(self):
        """A1: список должен быть отсортирован по производителю"""
        result = meow2.task_a1(meow2.producers, meow2.details)
        producers_list = [p for _, _, p in result]
        self.assertEqual(producers_list, sorted(producers_list))

    def test_task_a2_sum_for_zavod_detalei(self):
        """A2: проверяем суммарную стоимость для 'Завод деталей'"""
        result = meow2.task_a2(meow2.producers, meow2.details)
        target = next(r for r in result if r[0] == "Завод деталей")
        # 10+8=18
        self.assertEqual(target[1], 18)

    def test_task_a3_only_zavod(self):
        """A3: должны быть выбраны только производители, содержащие слово 'завод'"""
        result = meow2.task_a3(meow2.producers, meow2.details, meow2.details_producers)
        expected = {
            "Завод деталей",
```

```
            "Машиностроительный завод",
            "Завод МеталлПром",
        }
        self.assertEqual(set(result.keys()), expected)
    def test_task_a3_details_not_empty(self):
        """A3: у каждого завода должен быть хотя бы один элемент"""
        result = meow2.task_a3(meow2.producers, meow2.details,
meow2.details_producers)
        for key, detail_list in result.items():
            self.assertGreater(len(detail_list), 0)

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

test_task_a1_sorted_by_producer: проверяет, что список для задания A1 корректно отсортирован по имени производителя; test_task_a2_sum_for_zavod_detalei: проверяет правильность вычисления суммарной стоимости деталей у производителя «Завод деталей»; test_task_a3_only_zavod: удостоверяется, что в выборку A3 попадают только производители, содержащие слово «завод»; test_task_a3_details_not_empty: проверяет, что у всех производителей из задания A3 есть связанные детали.

Результат:

```
----------------------

Ran 4 tests in 0.001s


OK
```