Постановка задачи

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 pprint import pprint

from models import Book, Bookstore, BookBookstore
from src import (
    build_store_dict,
    build_book_dict,
    get_one_to_many,
    get_many_to_many,
    get_books_starting_with_a,
    get_min_price_per_store,
    get_books_with_stores
)


def main():
    bookstores = [
        Bookstore(1, 'Книжный мир'),
        Bookstore(2, 'Буквоед'),
        Bookstore(3, 'Азбука-Аттикус'),
        Bookstore(4, 'Читай-город'),
        Bookstore(5, 'Дом книги'),
    ]

    books = [
        Book(1, 'Анна Каренина', 500.0, 1),
        Book(2, 'Преступление и наказание', 600.0, 2),
        Book(3, 'Алые паруса', 450.0, 3),
        Book(4, 'Мастер и Маргарита', 550.0, 2),
        Book(5, 'Алиса в стране чудес', 700.0, 1),
    ]

    book_bookstores = [
        BookBookstore(1, 1),
        BookBookstore(2, 2),
        BookBookstore(3, 3),
        BookBookstore(4, 2),
        BookBookstore(5, 1),
        BookBookstore(1, 3),
        BookBookstore(3, 2),
        BookBookstore(5, 2),
    ]
```

```
    store_dict = build_store_dict(bookstores)
    book_dict = build_book_dict(books)

    one_to_many = get_one_to_many(books, store_dict)

    many_to_many = get_many_to_many(book_bookstores, book_dict, store_dict)

    print('Задание B1')
    res_1 = get_books_starting_with_a(one_to_many)
    pprint(res_1)

    print('\nЗадание B2')
    res_2 = get_min_price_per_store(one_to_many)
    pprint(res_2, width=60)

    print('\nЗадание B3')
    res_3 = get_books_with_stores(many_to_many)
    pprint(res_3)


if __name__ == '__main__':
    main()

/

from dataclasses import dataclass


@dataclass(order=True)
class Book:
    """
    Книга
    """
    id: int
    title: str
    price: float
    store_id: int

    @property
    def first_letter(self) -> str:
        return self.title[0]


@dataclass(order=True)
class Bookstore:
    """
    Книжный магазин
    """
    id: int
    name: str


@dataclass
```

```
class BookBookstore:
    """
    Связи многие-ко-многим
    """
    book_id: int
    store_id: int

/

from typing import List, Dict, Tuple
from itertools import groupby

from models import Book, Bookstore, BookBookstore


def build_store_dict(bookstores: List[Bookstore]) -> Dict[int, Bookstore]:
    return {store.id: store for store in bookstores}


def build_book_dict(books: List[Book]) -> Dict[int, Book]:
    return {book.id: book for book in books}


def get_one_to_many(
        books: List[Book],
        store_dict: Dict[int, Bookstore]
) -> List[Tuple[Book, Bookstore]]:
    return [
        (book, store_dict[book.store_id])
        for book in books
        if book.store_id in store_dict
    ]


def get_many_to_many(
        book_bookstores: List[BookBookstore],
        book_dict: Dict[int, Book],
        store_dict: Dict[int, Bookstore]
) -> List[Tuple[Book, Bookstore]]:
    return [
        (book_dict[bb.book_id], store_dict[bb.store_id])
        for bb in book_bookstores
        if bb.book_id in book_dict and bb.store_id in store_dict
    ]


def get_books_starting_with_a(one_to_many: List[Tuple[Book, Bookstore]]) ->
List[Tuple[str, str]]:
    """
    Задание B1:
    Получает список (название книги, название магазина) для книг, начинающихся на 'A'.
    """
    return [
        (book.title, store.name)
```

```
        for book, store in one_to_many
        if book.first_letter == 'A'
    ]


def get_min_price_per_store(one_to_many: List[Tuple[Book, Bookstore]]) -> List[Tuple[str,
float]]:
    """
    Задание B2:
    Получает список (название магазина, минимальная цена книги в этом магазине),
    отсортированный по возрастанию минимальной цены.
    """

    sorted_one_to_many = sorted(one_to_many, key=lambda x: x[1].name)
    grouped = groupby(sorted_one_to_many, key=lambda x: x[1].name)

    result = [
        (store_name, min(book.price for book, _ in books_in_store))
        for store_name, books_in_store in grouped
    ]

    return sorted(result, key=lambda x: x[1])


def get_books_with_stores(
        many_to_many: List[Tuple[Book, Bookstore]]
) -> List[Tuple[str, float, str]]:
    """
    Задание B3:
    Получает список (название книги, цена, название магазина),
    отсортированный по названию книги.
    """
    return [
        (book.title, book.price, store.name)
        for book, store in sorted(many_to_many, key=lambda x: x[0].title)
    ]

/

import unittest
from models import Book, Bookstore, BookBookstore
from src import (
    build_store_dict,
    build_book_dict,
    get_one_to_many,
    get_many_to_many,
    get_books_starting_with_a,
    get_min_price_per_store,
    get_books_with_stores
)


class TestLogic(unittest.TestCase):
    def setUp(self):
```

```
        self.bookstores = [
            Bookstore(1, 'Книжный мир'),
            Bookstore(2, 'Буквоед'),
            Bookstore(3, 'Азбука-Аттикус'),
            Bookstore(4, 'Читай-город'),
            Bookstore(5, 'Дом книги'),
        ]

        self.books = [
            Book(1, 'Анна Каренина', 500.0, 1),
            Book(2, 'Преступление и наказание', 600.0, 2),
            Book(3, 'Алые паруса', 450.0, 3),
            Book(4, 'Мастер и Маргарита', 550.0, 2),
            Book(5, 'Алиса в стране чудес', 700.0, 1),
        ]

        self.book_bookstores = [
            BookBookstore(1, 1),
            BookBookstore(2, 2),
            BookBookstore(3, 3),
            BookBookstore(4, 2),
            BookBookstore(5, 1),
            BookBookstore(1, 3),
            BookBookstore(3, 2),
            BookBookstore(5, 2),
        ]

        self.store_dict = build_store_dict(self.bookstores)
        self.book_dict = build_book_dict(self.books)
        self.one_to_many = get_one_to_many(self.books, self.store_dict)
        self.many_to_many = get_many_to_many(self.book_bookstores, self.book_dict,
self.store_dict)

    # -------------------------------------------------------------------------
    # Тесты для build_store_dict, build_book_dict
    # -------------------------------------------------------------------------
    def test_build_store_dict_keys(self):
        expected_keys = {1, 2, 3, 4, 5}
        self.assertEqual(set(self.store_dict.keys()), expected_keys)
        self.assertIsInstance(self.store_dict[1], Bookstore)

    def test_build_book_dict_keys(self):
        expected_keys = {1, 2, 3, 4, 5}
        self.assertEqual(set(self.book_dict.keys()), expected_keys)
        self.assertIsInstance(self.book_dict[1], Book)

    # -------------------------------------------------------------------------
    # Тесты для get_one_to_many
    # -------------------------------------------------------------------------
    def test_get_one_to_many_result_length(self):
        self.assertEqual(len(self.one_to_many), 5)

    def test_get_one_to_many_content(self):
        for book, store in self.one_to_many:
```

```
            self.assertIsInstance(book, Book)
            self.assertIsInstance(store, Bookstore)
            self.assertEqual(book.store_id, store.id)

    def test_get_one_to_many_empty_lists(self):
        result = get_one_to_many([], {})
        self.assertEqual(result, [])

    # ------------------------------------------------------------------------
    # Тесты для get_many_to_many
    # ------------------------------------------------------------------------
    def test_get_many_to_many_result_length(self):
        self.assertEqual(len(self.many_to_many), 8)

    def test_get_many_to_many_content(self):
        for book, store in self.many_to_many:
            self.assertIsInstance(book, Book)
            self.assertIsInstance(store, Bookstore)

    def test_get_many_to_many_empty(self):
        result = get_many_to_many([], self.book_dict, self.store_dict)
        self.assertEqual(result, [])

    # ------------------------------------------------------------------------
    # Тесты для get_books_starting_with_a (Задание B1)
    # ------------------------------------------------------------------------
    def test_books_starting_with_a_common(self):
        result = get_books_starting_with_a(self.one_to_many)

        self.assertIn(('Анна Каренина', 'Книжный мир'), result)
        self.assertIn(('Алиса в стране чудес', 'Книжный мир'), result)
        self.assertIn(('Алые паруса', 'Азбука-Аттикус'), result)

        for book_title, _ in result:
            self.assertTrue(book_title.startswith('A'))

    def test_books_starting_with_a_case_sensitivity(self):
        books_temp = [
            Book(6, 'anna', 111.0, 1),
            Book(7, 'Альбом', 222.0, 1),
        ]

        one_to_many_temp = get_one_to_many(books_temp, self.store_dict)
        result = get_books_starting_with_a(one_to_many_temp)

        self.assertIn(('Альбом', 'Книжный мир'), result)
        self.assertNotIn(('anna', 'Книжный мир'), result)

    def test_books_starting_with_a_empty_input(self):
        result = get_books_starting_with_a([])
        self.assertEqual(result, [])

    # ------------------------------------------------------------------------
    # Тесты для get_min_price_per_store (Задание B2)
```

```
    # ------------------------------------------------------------------------
    def test_min_price_per_store_common(self):
        result = get_min_price_per_store(self.one_to_many)

        expected = [
            ('Азбука-Аттикус', 450.0),
            ('Книжный мир', 500.0),
            ('Буквоед', 550.0),
        ]
        self.assertEqual(result, expected)

    def test_min_price_per_store_sorting(self):
        result = get_min_price_per_store(self.one_to_many)
        prices = [item[1] for item in result]
        self.assertEqual(prices, sorted(prices))

    def test_min_price_per_store_empty_input(self):
        result = get_min_price_per_store([])
        self.assertEqual(result, [])

    # ------------------------------------------------------------------------
    # Тесты для get_books_with_stores (Задание B3)
    # ------------------------------------------------------------------------
    def test_books_with_stores_common(self):
        result = get_books_with_stores(self.many_to_many)

        self.assertIn(('Анна Каренина', 500.0, 'Книжный мир'), result)
        self.assertIn(('Анна Каренина', 500.0, 'Азбука-Аттикус'), result)

        self.assertIn(('Алиса в стране чудес', 700.0, 'Книжный мир'), result)
        self.assertIn(('Алиса в стране чудес', 700.0, 'Буквоед'), result)

    def test_books_with_stores_sorting(self):
        result = get_books_with_stores(self.many_to_many)

        titles = [item[0] for item in result]

        self.assertEqual(titles, sorted(titles))

    def test_books_with_stores_empty_input(self):
        result = get_books_with_stores([])
        self.assertEqual(result, [])

    def test_books_with_stores_duplicate_check(self):
        duplicated_book_bookstores = self.book_bookstores + [
            BookBookstore(1, 1),
            BookBookstore(3, 2),
        ]

        many_to_many_dups = get_many_to_many(duplicated_book_bookstores, self.book_dict,
self.store_dict)
        result = get_books_with_stores(many_to_many_dups)

        from collections import Counter
```

```
        counter_result = Counter(result)
        self.assertGreaterEqual(
            counter_result[('Анна Каренина', 500.0, 'Книжный мир')],
            2
        )
        self.assertGreaterEqual(
            counter_result[('Алые паруса', 450.0, 'Буквоед')],
            2
        )


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

Анализ результатов