МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Дисциплина “Парадигмы и конструкции языков программирования”

Отчет по рубежному контролю №2

Выполнил:
Студент группы ИУ5Ц-54Б
Цурин А.П.
Преподаватель:
Гапанюк Ю.Е.

Москва 2025

1. Задания для выполнения

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).

2. Листинг программы

```
from operator import itemgetter
import unittest

class DataRow:
    """Строка данных"""
    def __init__(self, id, name, value, table_id):
        self.id = id
        self.name = name  # название строки
        self.value = value  # числовое значение (аналог зарплаты)
        self.table_id = table_id

    def __repr__(self):
        return f"DataRow(id={self.id}, name='{self.name}', value={self.value},
table_id={self.table_id})"

class DataTable:
    """Таблица данных"""
    def __init__(self, id, name):
        self.id = id
        self.name = name  # название таблицы

    def __repr__(self):
        return f"DataTable(id={self.id}, name='{self.name}')"

class RowTable:
    """Строки таблиц для связи многие-ко-многим"""
    def __init__(self, row_id, table_id):
        self.row_id = row_id
        self.table_id = table_id

    def __repr__(self):
        return f"RowTable(row_id={self.row_id}, table_id={self.table_id})"


class DataProcessor:
    """Класс для обработки данных с тестовыми данными по умолчанию"""

    def __init__(self, tables=None, rows=None, row_table=None):
        """Инициализация с возможностью передачи тестовых данных"""
        self.tables = tables if tables is not None else self._get_default_tables()
        self.rows = rows if rows is not None else self._get_default_rows()
        self.row_table = row_table if row_table is not None else
self._get_default_row_table()
```

```python
    @staticmethod
    def _get_default_tables():
        return [
            DataTable(1, "Анализ продаж"),
            DataTable(2, "Бюджет"),
            DataTable(3, "Аудит качества"),
            DataTable(4, "Отчетность"),
            DataTable(5, "Анализ рисков")
        ]

    @staticmethod
    def _get_default_rows():
        return [
            DataRow(1, "Иванов", 50000, 1),
            DataRow(2, "Петров", 45000, 1),
            DataRow(3, "Сидоров", 30000, 2),
            DataRow(4, "Кузнецов", 60000, 3),
            DataRow(5, "Александров", 35000, 3),
            DataRow(6, "Сергеев", 55000, 4),
            DataRow(7, "Антонов", 40000, 5),
            DataRow(8, "Николаев", 48000, 5)
        ]

    @staticmethod
    def _get_default_row_table():
        return [
            RowTable(1, 1),
            RowTable(2, 1),
            RowTable(3, 2),
            RowTable(4, 3),
            RowTable(5, 3),
            RowTable(6, 4),
            RowTable(7, 5),
            RowTable(8, 5),
            RowTable(1, 3),  # дополнительная связь для многих-ко-многим
            RowTable(4, 5)   # дополнительная связь для многих-ко-многим
        ]

    def get_one_to_many(self):
        """Соединение данных один-ко-многим"""
        return [(r.name, r.value, t.name)
                for t in self.tables
                for r in self.rows
                if r.table_id == t.id]

    def get_many_to_many(self):
        """Соединение данных многие-ко-многим"""
        many_to_many_temp = [(t.name, rt.table_id, rt.row_id)
                             for t in self.tables
                             for rt in self.row_table
                             if t.id == rt.table_id]

        return [(r.name, r.value, table_name)
```

```
            for table_name, table_id, row_id in many_to_many_temp
            for r in self.rows if r.id == row_id]

def task1_rows_ending_with_ov(self):
    """Задание 1: Строки, оканчивающиеся на 'ов', и их таблицы"""
    if not self.rows or not self.tables:
        return []

    one_to_many = self.get_one_to_many()
    return [(row_name, table_name)
            for row_name, _, table_name in one_to_many
            if row_name.endswith("ов")]

def task2_tables_with_avg_value(self):
    """Задание 2: Таблицы со средним значением строк (отсортировано)"""
    if not self.rows or not self.tables:
        return []

    one_to_many = self.get_one_to_many()

    # Создаем словарь для хранения сумм и количеств
    table_stats = {}
    for row_name, value, table_name in one_to_many:
        if table_name not in table_stats:
            table_stats[table_name] = {'sum': 0, 'count': 0}
        table_stats[table_name]['sum'] += value
        table_stats[table_name]['count'] += 1

    # Вычисляем средние значения
    result = []
    for table_name, stats in table_stats.items():
        avg_value = stats['sum'] / stats['count']
        result.append((table_name, avg_value))

    # Сортируем по среднему значению
    return sorted(result, key=itemgetter(1))

def task3_tables_starting_with_a(self):
    """Задание 3: Таблицы, начинающиеся на 'A', и их строки"""
    if not self.rows or not self.tables:
        return {}

    many_to_many = self.get_many_to_many()

    # Фильтруем таблицы, начинающиеся на "A"
    a_table_names = [t.name for t in self.tables if t.name.startswith("A")]

    # Для каждой такой таблицы находим связанные строки
    result = {}
    for table_name in a_table_names:
        # Фильтруем строки для текущей таблицы
        table_rows = list(filter(lambda x: x[2] == table_name, many_to_many))
        # Убираем дубликаты по имени строки
```

```
            unique_rows = []
            seen_names = set()
            for row in table_rows:
                row_name, row_value, _ = row
                if row_name not in seen_names:
                    unique_rows.append((row_name, row_value))
                    seen_names.add(row_name)
            if unique_rows:  # Добавляем только если есть строки
                result[table_name] = unique_rows

        return result


def main():
    """Основная функция для вывода результатов"""
    processor = DataProcessor()

    print("=" * 60)
    print("Задание 1: Строки, оканчивающиеся на 'ов', и их таблицы")
    print("=" * 60)
    res_1 = processor.task1_rows_ending_with_ov()
    for row_name, table_name in res_1:
        print(f"Строка: {row_name}, Таблица: {table_name}")
    print()

    print("=" * 60)
    print("Задание 2: Таблицы со средним значением строк (отсортировано)")
    print("=" * 60)
    res_2 = processor.task2_tables_with_avg_value()
    for table_name, avg_value in res_2:
        print(f"Таблица: {table_name}, Среднее значение: {avg_value:.2f}")
    print()

    print("=" * 60)
    print("Задание 3: Таблицы, начинающиеся на 'A', и их строки")
    print("=" * 60)
    res_3 = processor.task3_tables_starting_with_a()
    for table_name, rows_list in res_3.items():
        print(f"\nТаблица: {table_name}")
        for row_name, row_value in rows_list:
            print(f"  - {row_name}: {row_value}")


# ============================================================================
# МОДУЛЬНЫЕ ТЕСТЫ
# ============================================================================

class TestDataProcessor(unittest.TestCase):

    def setUp(self):
        """Настройка тестовых данных"""
        # Тестовые таблицы
        self.test_tables = [
```

```
            DataTable(1, "Анализ продаж"),
            DataTable(2, "Бюджет"),
            DataTable(3, "Аудит качества")
        ]

        # Тестовые строки
        self.test_rows = [
            DataRow(1, "Иванов", 50000, 1),
            DataRow(2, "Петров", 45000, 1),
            DataRow(3, "Сидоров", 30000, 2),
            DataRow(4, "Кузнецов", 60000, 3),
            DataRow(5, "Александров", 35000, 3)
        ]

        # Тестовые связи многие-ко-многим
        self.test_row_table = [
            RowTable(1, 1),
            RowTable(2, 1),
            RowTable(3, 2),
            RowTable(4, 3),
            RowTable(5, 3),
            RowTable(1, 3),  # дополнительная связь
        ]

        self.processor = DataProcessor(self.test_tables, self.test_rows,
self.test_row_table)

    def test_task1_rows_ending_with_ov(self):
        """Тест 1: строки, оканчивающиеся на 'ов'"""
        print("\n[Запуск теста 1: task1_rows_ending_with_ov]")
        result = self.processor.task1_rows_ending_with_ov()

        # Проверяем количество результатов
        self.assertEqual(len(result), 5, "Должно быть 5 строк, оканчивающихся на 'ов'")

        # Проверяем конкретные значения
        expected = [
            ("Иванов", "Анализ продаж"),
            ("Петров", "Анализ продаж"),
            ("Сидоров", "Бюджет"),
            ("Кузнецов", "Аудит качества"),
            ("Александров", "Аудит качества")
        ]

        for expected_row in expected:
            self.assertIn(expected_row, result, f"Ожидалось: {expected_row}")

        # Проверяем, что строки без "ов" не попадают в результат
        test_rows_without_ov = self.test_rows + [DataRow(6, "Смирнов", 40000, 1)]
        processor_without_ov = DataProcessor(self.test_tables, test_rows_without_ov,
self.test_row_table)
        result_without_ov = processor_without_ov.task1_rows_ending_with_ov()
```

```
        self.assertEqual(len(result_without_ov), 6, "Должно быть 6 строк после
добавления 'Смирнов'")

        print("Тест 1 пройден успешно")

    def test_task2_tables_with_avg_value(self):
        """Тест 2: таблицы со средним значением"""
        print("\n[Запуск теста 2: task2_tables_with_avg_value]")
        result = self.processor.task2_tables_with_avg_value()

        # Проверяем количество таблиц
        self.assertEqual(len(result), 3, "Должно быть 3 таблицы")

        # Проверяем порядок сортировки (по возрастанию среднего)
        avg_values = [avg for _, avg in result]
        self.assertEqual(avg_values, sorted(avg_values), "Средние значения должны быть
отсортированы")

        # Проверяем вычисление средних значений
        expected_avg = {
            "Бюджет": 30000.0,  # 30000 / 1
            "Аудит качества": 47500.0,  # (60000 + 35000) / 2
            "Анализ продаж": 47500.0  # (50000 + 45000) / 2
        }

        for table_name, avg_value in result:
            self.assertAlmostEqual(avg_value, expected_avg[table_name], places=2,
                                   msg=f"Неверное среднее для таблицы {table_name}")

        print("Тест 2 пройден успешно")

    def test_task3_tables_starting_with_a(self):
        """Тест 3: таблицы, начинающиеся на 'А'"""
        print("\n[Запуск теста 3: task3_tables_starting_with_a]")
        result = self.processor.task3_tables_starting_with_a()

        # Проверяем количество таблиц, начинающихся на "А"
        self.assertEqual(len(result), 2, "Должно быть 2 таблицы, начинающихся на 'А'")

        # Проверяем наличие нужных таблиц
        self.assertIn("Анализ продаж", result, "Таблица 'Анализ продаж' должна быть в
результате")
        self.assertIn("Аудит качества", result, "Таблица 'Аудит качества' должна быть в
результате")
        self.assertNotIn("Бюджет", result, "Таблица 'Бюджет' не должна быть в
результате")

        # Проверяем строки для таблицы "Анализ продаж"
        sales_rows = result["Анализ продаж"]
        self.assertEqual(len(sales_rows), 2, "В таблице 'Анализ продаж' должно быть 2
строки")
        self.assertIn(("Иванов", 50000), sales_rows, "Строка 'Иванов' должна быть в
таблице")
```

```
        self.assertIn(("Петров", 45000), sales_rows, "Строка 'Петров' должна быть в
таблице")

        # Проверяем строки для таблицы "Аудит качества"
        quality_rows = result["Аудит качества"]
        self.assertEqual(len(quality_rows), 3, "В таблице 'Аудит качества' должно быть
3 строки (Кузнецов, Александров, Иванов)")
        self.assertIn(("Кузнецов", 60000), quality_rows, "Строка 'Кузнецов' должна быть
в таблице")
        self.assertIn(("Александров", 35000), quality_rows, "Строка 'Александров'
должна быть в таблице")
        self.assertIn(("Иванов", 50000), quality_rows, "Строка 'Иванов' должна быть в
таблице (связь многие-ко-многим)")

        print("Тест 3 пройден успешно")

    def test_edge_cases(self):
        """Тест граничных случаев"""
        print("\n[Запуск теста: edge_cases]")

        # Тест с пустыми данными - явно передаем пустые списки
        empty_processor = DataProcessor([], [], [])

        # Задание 1 с пустыми данными
        result1 = empty_processor.task1_rows_ending_with_ov()
        self.assertEqual(result1, [], "При пустых данных должен возвращаться пустой
список")

        # Задание 2 с пустыми данными
        result2 = empty_processor.task2_tables_with_avg_value()
        self.assertEqual(result2, [], "При пустых данных должен возвращаться пустой
список")

        # Задание 3 с пустыми данными
        result3 = empty_processor.task3_tables_starting_with_a()
        self.assertEqual(result3, {}, "При пустых данных должен возвращаться пустой
словарь")

        # Тест с таблицами без строк
        tables_only = [DataTable(1, "Анализ"), DataTable(2, "Бюджет")]
        processor_tables_only = DataProcessor(tables_only, [], [])
        result3_empty = processor_tables_only.task3_tables_starting_with_a()
        self.assertEqual(result3_empty, {}, "При таблицах без строк должен возвращаться
пустой словарь")

        # Тест с строками без таблиц
        rows_only = [DataRow(1, "Иванов", 50000, 1)]
        processor_rows_only = DataProcessor([], rows_only, [])
        result1_rows_only = processor_rows_only.task1_rows_ending_with_ov()
        self.assertEqual(result1_rows_only, [], "При отсутствии таблиц должен
возвращаться пустой список")

        print("Тест граничных случаев пройден успешно")
```

```
    def test_one_to_many_relationship(self):
        """Тест связи один-ко-многим"""
        print("\n[Запуск теста: one_to_many_relationship]")
        one_to_many = self.processor.get_one_to_many()

        # Проверяем структуру данных
        for row_name, value, table_name in one_to_many:
            self.assertIsInstance(row_name, str, "Имя строки должно быть строкой")
            self.assertIsInstance(value, (int, float), "Значение должно быть числом")
            self.assertIsInstance(table_name, str, "Имя таблицы должно быть строкой")

        # Проверяем правильность связей
        row_1_connections = [(rn, tn) for rn, _, tn in one_to_many if rn == "Иванов"]
        self.assertIn(("Иванов", "Анализ продаж"), row_1_connections,
                      "Строка 'Иванов' должна быть связана с таблицей 'Анализ продаж'")

        print("Тест связи один-ко-многим пройден успешно")

    def test_many_to_many_relationship(self):
        """Тест связи многие-ко-многим"""
        print("\n[Запуск теста: many_to_many_relationship]")
        many_to_many = self.processor.get_many_to_many()

        # Проверяем, что Иванов появляется в двух таблицах
        ivanov_tables = [tn for rn, _, tn in many_to_many if rn == "Иванов"]
        self.assertIn("Анализ продаж", ivanov_tables,
                      "Иванов должен быть в таблице 'Анализ продаж'")
        self.assertIn("Аудит качества", ivanov_tables,
                      "Иванов должен быть в таблице 'Аудит качества' (связь многие-ко-
многим)")

        print("Тест связи многие-ко-многим пройден успешно")


# ============================================================================
# ТОЧКА ВХОДА
# ============================================================================

if __name__ == "__main__":
    print("=" * 60)
    print("РУБЕЖНЫЙ КОНТРОЛЬ №2")
    print("Рефакторинг и модульное тестирование")
    print("=" * 60)

    # Спросим пользователя, что запускать
    choice = input("\nВыберите действие:\n"
                   "1 - Запустить основную программу\n"
                   "2 - Запустить тесты\n"
                   "3 - Запустить всё\n"
                   "Ваш выбор (1-3): ").strip()

    if choice == "1":
```

```
        print("\n" + "=" * 60)
        print("ЗАПУСК ОСНОВНОЙ ПРОГРАММЫ")
        print("=" * 60)
        main()

    elif choice == "2":
        print("\n" + "=" * 60)
        print("ЗАПУСК МОДУЛЬНЫХ ТЕСТОВ")
        print("=" * 60)
        # Создаем test suite и запускаем
        suite = unittest.TestLoader().loadTestsFromTestCase(TestDataProcessor)
        runner = unittest.TextTestRunner(verbosity=2)
        runner.run(suite)

    elif choice == "3":
        print("\n" + "=" * 60)
        print("ЗАПУСК ТЕСТОВ")
        print("=" * 60)
        # Запускаем тесты
        suite = unittest.TestLoader().loadTestsFromTestCase(TestDataProcessor)
        runner = unittest.TextTestRunner(verbosity=2)
        test_result = runner.run(suite)

        print("\n" + "=" * 60)
        print("ЗАПУСК ОСНОВНОЙ ПРОГРАММЫ")
        print("=" * 60)
        # Запускаем основную программу
        main()

    else:
        print("Неверный выбор. Запускаю тесты по умолчанию...")
        suite = unittest.TestLoader().loadTestsFromTestCase(TestDataProcessor)
        runner = unittest.TextTestRunner(verbosity=2)
        runner.run(suite)
```

3. Результаты работы программы

```
PS C:\Users\student\project> python .\РК2.py
============================================================
РУБЕЖНЫЙ КОНТРОЛЬ №2
Рефакторинг и модульное тестирование
============================================================

Выберите действие:
1 - Запустить основную программу
2 - Запустить тесты
3 - Запустить всё
Ваш выбор (1-3): 3

============================================================
ЗАПУСК ТЕСТОВ
============================================================
test_edge_cases (__main__.TestDataProcessor.test_edge_cases)
```

Тест граничных случаев ...
[Запуск теста: edge_cases]
Тест граничных случаев пройден успешно
ok
test_many_to_many_relationship (__main__.TestDataProcessor.test_many_to_many_relationship)
Тест связи многие-ко-многим ...
[Запуск теста: many_to_many_relationship]
Тест связи многие-ко-многим пройден успешно
ok
test_one_to_many_relationship (__main__.TestDataProcessor.test_one_to_many_relationship)
Тест связи один-ко-многим ...
[Запуск теста: one_to_many_relationship]
Тест связи один-ко-многим пройден успешно
ok
test_task1_rows_ending_with_ov (__main__.TestDataProcessor.test_task1_rows_ending_with_ov)
Тест 1: строки, оканчивающиеся на 'ов' ...
[Запуск теста 1: task1_rows_ending_with_ov]
Тест 1 пройден успешно
ok
test_task2_tables_with_avg_value (__main__.TestDataProcessor.test_task2_tables_with_avg_value)
Тест 2: таблицы со средним значением ...
[Запуск теста 2: task2_tables_with_avg_value]
Тест 2 пройден успешно
ok
test_task3_tables_starting_with_a (__main__.TestDataProcessor.test_task3_tables_starting_with_a)
Тест 3: таблицы, начинающиеся на 'А' ...
[Запуск теста 3: task3_tables_starting_with_a]
Тест 3 пройден успешно
ok

Ran 6 tests in 0.004s

OK

==========
ЗАПУСК ОСНОВНОЙ ПРОГРАММЫ
==========
==========
Задание 1: Строки, оканчивающиеся на 'ов', и их таблицы
==========
Строка: Иванов, Таблица: Анализ продаж
Строка: Петров, Таблица: Анализ продаж
Строка: Сидоров, Таблица: Бюджет
Строка: Кузнецов, Таблица: Аудит качества
Строка: Александров, Таблица: Аудит качества
Строка: Антонов, Таблица: Анализ рисков

==========
Задание 2: Таблицы со средним значением строк (отсортировано)
==========
Таблица: Бюджет, Среднее значение: 30000.00
Таблица: Анализ рисков, Среднее значение: 44000.00
Таблица: Анализ продаж, Среднее значение: 47500.00
Таблица: Аудит качества, Среднее значение: 47500.00

Таблица: Отчетность, Среднее значение: 55000.00

==========
Задание 3: Таблицы, начинающиеся на 'А', и их строки
==========

Таблица: Анализ продаж
- Иванов: 50000
- Петров: 45000

Таблица: Аудит качества
- Кузнецов: 60000
- Александров: 35000
- Иванов: 50000

Таблица: Анализ рисков
- Антонов: 40000
- Николаев: 48000
- Кузнецов: 60000