Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по рубежному контролю №2

«**Вариант А, 23**»

Выполнил:
Студент группы ИУ5-31Б
Шанурина Екатерина

Проверил:
Гапанюк Ю.Е.

2025 г.

Листинг кода

Main.py

```
from operator import itemgetter


class SyntaxConstruct:
    """Синтаксическая конструкция"""
    def __init__(self, id, name, complexity, lang_id):
        self.id = id
        self.name = name
        self.complexity = complexity
        self.lang_id = lang_id


class ProgrammingLanguage:
    """Язык программирования"""
    def __init__(self, id, name):
        self.id = id
        self.name = name


class ConstructLanguage:
    """Синтаксические конструкции языков программирования (для связи многие-ко-многим)"""
    def __init__(self, lang_id, construct_id):
        self.lang_id = lang_id
        self.construct_id = construct_id


class LanguageDataProcessor:
    """Класс для обработки данных о языках программирования и синтаксических конструкциях"""

    def __init__(self, langs, constructs, constructs_langs):
        self.langs = langs
        self.constructs = constructs
        self.constructs_langs = constructs_langs

    def get_one_to_many(self):
        """Получение связи один-ко-многим (язык → конструкции)"""
        return [(c.name, c.complexity, l.name)
                for l in self.langs
                for c in self.constructs
                if c.lang_id == l.id]

    def get_many_to_many(self):
        """Получение связи многие-ко-многим"""

        many_to_many_temp = [(l.name, cl.lang_id, cl.construct_id)
```

```
                                for l in self.langs
                                for cl in self.constructs_langs
                                if l.id == cl.lang_id]


        return [(c.name, c.complexity, lang_name)
                for lang_name, lang_id, construct_id in many_to_many_temp
                for c in self.constructs
                if c.id == construct_id]

    def task_a1(self):
        """Задание A1: Список всех связанных конструкций и языков,
отсортированный по языкам"""
        one_to_many = self.get_one_to_many()
        return sorted(one_to_many, key=itemgetter(2))

    def task_a2(self):
        """Задание A2: Список языков с суммарной сложностью конструкций,
отсортированный по сложности"""
        one_to_many = self.get_one_to_many()
        result = []

        for lang in self.langs:

            lang_constructs = list(filter(lambda item: item[2] == lang.name,
one_to_many))

            if len(lang_constructs) > 0:

                complexities = [complexity for _, complexity, _ in
lang_constructs]

                total_complexity = sum(complexities)
                result.append((lang.name, total_complexity))


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

    def task_a3(self):
        """Задание A3: Языки со словом 'Java' и их конструкции"""
        many_to_many = self.get_many_to_many()
        result = []

        for lang in self.langs:
            if 'Java' in lang.name:

                lang_constructs = list(filter(lambda item: item[2] == lang.name,
many_to_many))

                construct_names = [name for name, _, _ in lang_constructs]
```

```
                for construct_name in construct_names:
                    result.append((lang.name, construct_name))

        return result

    def get_language_by_name(self, name):
        """Получить язык программирования по имени"""
        for lang in self.langs:
            if lang.name == name:
                return lang
        return None

    def get_construct_by_name(self, name):
        """Получить синтаксическую конструкцию по имени"""
        for construct in self.constructs:
            if construct.name == name:
                return construct
        return None


def create_test_data():
    """Создание тестовых данных"""

    langs = [
        ProgrammingLanguage(1, 'Python'),
        ProgrammingLanguage(2, 'JavaScript'),
        ProgrammingLanguage(3, 'C++'),
        ProgrammingLanguage(11, 'Java'),
        ProgrammingLanguage(22, 'Go'),
    ]


    constructs = [
        SyntaxConstruct(1, 'List comprehension', 7, 1),
        SyntaxConstruct(2, 'Arrow function', 5, 2),
        SyntaxConstruct(3, 'Template', 8, 3),
        SyntaxConstruct(4, 'Lambda expression', 6, 2),
        SyntaxConstruct(5, 'Decorator', 9, 1),
    ]


    constructs_langs = [
        ConstructLanguage(1, 1),
        ConstructLanguage(2, 2),
        ConstructLanguage(3, 3),
        ConstructLanguage(2, 4),
        ConstructLanguage(1, 5),
        ConstructLanguage(11, 4),
        ConstructLanguage(22, 2),
    ]
```

```
    return langs, constructs, constructs_langs


def print_table(headers, data):
    """Функция для форматированного вывода таблицы"""
    if not data:
        print("Нет данных для отображения")
        return


    col_widths = []
    for i in range(len(headers)):
        max_len = len(headers[i])
        for row in data:
            cell_len = len(str(row[i]))
            if cell_len > max_len:
                max_len = cell_len
        col_widths.append(max_len)


    header_str = "  ".join(f"{headers[i]:<{col_widths[i]}}" for i in
range(len(headers)))
    print(header_str)


    separator = "  ".join("-" * col_widths[i] for i in range(len(headers)))
    print(separator)


    for row in data:
        row_str = "  ".join(f"{str(row[i]):<{col_widths[i]}}" for i in
range(len(row)))
        print(row_str)

    print()


def main():
    """Основная функция программы"""

    langs, constructs, constructs_langs = create_test_data()


    processor = LanguageDataProcessor(langs, constructs, constructs_langs)

    print('=' * 60)
    print('РУБЕЖНЫЙ КОНТРОЛЬ №1')
    print('Предметная область: Синтаксическая конструкция - Язык
программирования')
    print('=' * 60)
    print()
```

```
    print('Задание A1:')
    print('Список всех связанных конструкций и языков, отсортированный по
языкам')
    result_a1 = processor.task_a1()
    print_table(['Конструкция', 'Сложность', 'Язык'], result_a1)


    print('Задание A2:')
    print('Языки с суммарной сложностью конструкций (сортировка по убыванию)')
    result_a2 = processor.task_a2()
    print_table(['Язык', 'Суммарная сложность'], result_a2)

    # Выполнение задания A3
    print('Задание A3:')
    print('Языки со словом "Java" и их конструкции')
    result_a3 = processor.task_a3()
    print_table(['Язык', 'Конструкция'], result_a3)

    print('=' * 60)
    print('ВСЕ ЗАДАНИЯ ВЫПОЛНЕНЫ')
    print('=' * 60)


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

test_main.py

```
from operator import itemgetter
import unittest
from main import (
    SyntaxConstruct,
    ProgrammingLanguage,
    ConstructLanguage,
    LanguageDataProcessor,
    create_test_data
)


class TestLanguageDataProcessor(unittest.TestCase):
    """Тесты для класса LanguageDataProcessor"""

    def setUp(self):
        """Подготовка тестовых данных перед каждым тестом"""
        self.langs, self.constructs, self.constructs_langs = create_test_data()
        self.processor = LanguageDataProcessor(
            self.langs,
            self.constructs,
            self.constructs_langs
        )
```

```
    def test_task_a1_structure_and_sorting(self):
        """Тест структуры и сортировки задания A1"""
        result = self.processor.task_a1()

        # Проверяем, что результат не пустой
        self.assertGreater(len(result), 0, "Результат задания A1 пуст")

        # Проверяем структуру каждой записи
        for item in result:
            self.assertEqual(len(item), 3, f"Неверная структура записи: {item}")
            self.assertIsInstance(item[0], str, f"Название конструкции должно
быть строкой: {item[0]}")
            self.assertIsInstance(item[1], int, f"Сложность должна быть числом:
{item[1]}")
            self.assertIsInstance(item[2], str, f"Название языка должно быть
строкой: {item[2]}")

        # Проверяем сортировку по названию языка
        language_names = [item[2] for item in result]
        sorted_language_names = sorted(language_names)
        self.assertEqual(language_names, sorted_language_names,
                         "Результат не отсортирован по языкам")

    def test_task_a2_calculation_and_sorting(self):
        """Тест расчетов и сортировки задания A2"""
        result = self.processor.task_a2()

        # Проверяем, что результат не пустой
        self.assertGreater(len(result), 0, "Результат задания A2 пуст")

        # Проверяем структуру каждой записи
        for item in result:
            self.assertEqual(len(item), 2, f"Неверная структура записи: {item}")
            self.assertIsInstance(item[0], str, f"Название языка должно быть
строкой: {item[0]}")
            self.assertIsInstance(item[1], int, f"Суммарная сложность должна быть
числом: {item[1]}")

        # Проверяем правильность расчетов
        one_to_many = self.processor.get_one_to_many()
        for lang_name, total_complexity in result:
            # Находим все конструкции для данного языка
            lang_constructs = [item for item in one_to_many if item[2] ==
lang_name]

            # Вычисляем суммарную сложность
            calculated_sum = sum(item[1] for item in lang_constructs)

            # Проверяем совпадение
            self.assertEqual(total_complexity, calculated_sum,
                             f"Неверный расчет для языка {lang_name}: "
```

```
                            f"ожидалось {calculated_sum}, получено
{total_complexity}")

        # Проверяем сортировку по убыванию сложности
        complexities = [item[1] for item in result]
        for i in range(len(complexities) - 1):
             self.assertGreaterEqual(complexities[i], complexities[i + 1],
                                     "Результат не отсортирован по убыванию
сложности")

    def test_task_a3_filtering(self):
        """Тест фильтрации задания А3"""
        result = self.processor.task_a3()

        # Проверяем структуру каждой записи
        for item in result:
            self.assertEqual(len(item), 2, f"Неверная структура записи: {item}")
            lang_name, construct_name = item

            # Проверяем, что название языка содержит "Java"
            self.assertIn('Java', lang_name,
                          f"Язык {lang_name} не содержит 'Java'")

            self.assertIsInstance(lang_name, str, "Название языка должно быть
строкой")
            self.assertIsInstance(construct_name, str, "Название конструкции
должно быть строкой")

        # Проверяем, что для каждого языка с "Java" есть конструкции
        java_languages = [lang for lang in self.langs if 'Java' in lang.name]
        for java_lang in java_languages:
            # Ищем конструкции для этого языка в результате
            lang_constructs = [item[1] for item in result if item[0] ==
java_lang.name]
            self.assertGreater(len(lang_constructs), 0,
                               f"Для языка {java_lang.name} не найдено
конструкций")

    def test_one_to_many_relationship_integrity(self):
        """Тест целостности связи один-ко-многим"""
        one_to_many = self.processor.get_one_to_many()

        for construct_name, complexity, lang_name in one_to_many:
            # Находим соответствующий язык
            lang = next((l for l in self.langs if l.name == lang_name), None)
            self.assertIsNotNone(lang, f"Язык {lang_name} не найден")

            # Находим соответствующую конструкцию
            construct = next((c for c in self.constructs
                              if c.name == construct_name and c.complexity ==
complexity), None)
```

```
            self.assertIsNotNone(construct, f"Конструкция {construct_name} не
найдена")

            # Проверяем связь
            self.assertEqual(construct.lang_id, lang.id,
                             f"Несоответствие связи: конструкция {construct_name}
"
                             f"должна принадлежать языку {lang_name}")

    def test_many_to_many_relationship_integrity(self):
        """Тест целостности связи многие-ко-многим"""
        many_to_many = self.processor.get_many_to_many()

        for construct_name, complexity, lang_name in many_to_many:
            # Находим соответствующий язык
            lang = next((l for l in self.langs if l.name == lang_name), None)
            self.assertIsNotNone(lang, f"Язык {lang_name} не найден")

            # Находим соответствующую конструкцию
            construct = next((c for c in self.constructs
                              if c.name == construct_name and c.complexity ==
complexity), None)
            self.assertIsNotNone(construct, f"Конструкция {construct_name} не
найдена")

            # Проверяем связь в таблице constructs_langs
            connection = next((cl for cl in self.constructs_langs
                               if cl.lang_id == lang.id and cl.construct_id ==
construct.id), None)
            self.assertIsNotNone(connection,
                                 f"Отсутствует связь между языком {lang_name} "
                                 f"и конструкцией {construct_name}")


class TestDataClasses(unittest.TestCase):
    """Тесты для классов данных"""

    def test_syntax_construct_creation(self):
        """Тест создания объекта SyntaxConstruct"""
        construct = SyntaxConstruct(1, 'Lambda expression', 6, 2)

        self.assertEqual(construct.id, 1)
        self.assertEqual(construct.name, 'Lambda expression')
        self.assertEqual(construct.complexity, 6)
        self.assertEqual(construct.lang_id, 2)

        # Проверяем граничные значения сложности
        construct_low = SyntaxConstruct(2, 'Simple construct', 1, 1)
        construct_high = SyntaxConstruct(3, 'Complex construct', 10, 1)

        self.assertEqual(construct_low.complexity, 1)
```

```
        self.assertEqual(construct_high.complexity, 10)

    def test_programming_language_creation(self):
        """Тест создания объекта ProgrammingLanguage"""
        language = ProgrammingLanguage(1, 'Python')

        self.assertEqual(language.id, 1)
        self.assertEqual(language.name, 'Python')

        # Проверяем создание языка с пробелами в названии
        language_with_spaces = ProgrammingLanguage(2, 'C Sharp')
        self.assertEqual(language_with_spaces.name, 'C Sharp')

    def test_construct_language_creation(self):
        """Тест создания объекта ConstructLanguage"""
        connection = ConstructLanguage(1, 2)

        self.assertEqual(connection.lang_id, 1)
        self.assertEqual(connection.construct_id, 2)

        # Проверяем обратную связь
        reverse_connection = ConstructLanguage(2, 1)
        self.assertEqual(reverse_connection.lang_id, 2)
        self.assertEqual(reverse_connection.construct_id, 1)


class TestHelperMethods(unittest.TestCase):
    """Тесты вспомогательных методов"""

    def setUp(self):
        """Подготовка тестовых данных"""
        self.langs, self.constructs, self.constructs_langs = create_test_data()
        self.processor = LanguageDataProcessor(
            self.langs,
            self.constructs,
            self.constructs_langs
        )

    def test_get_language_by_name(self):
        """Тест метода get_language_by_name"""
        # Существующий язык
        python_lang = self.processor.get_language_by_name('Python')
        self.assertIsNotNone(python_lang)
        self.assertEqual(python_lang.name, 'Python')
        self.assertEqual(python_lang.id, 1)

        # Несуществующий язык
        nonexistent_lang = self.processor.get_language_by_name('Ruby')
        self.assertIsNone(nonexistent_lang)

    def test_get_construct_by_name(self):
```

```
        """Тест метода get_construct_by_name"""
        # Существующая конструкция
        lambda_construct = self.processor.get_construct_by_name('Lambda
expression')
        self.assertIsNotNone(lambda_construct)
        self.assertEqual(lambda_construct.name, 'Lambda expression')
        self.assertEqual(lambda_construct.complexity, 6)

        # Несуществующая конструкция
        nonexistent_construct = self.processor.get_construct_by_name('Magic
function')
        self.assertIsNone(nonexistent_construct)


def run_all_tests():
    """Запуск всех тестов с подробным выводом"""
    # Создаем тестовый набор
    loader = unittest.TestLoader()
    suite = unittest.TestSuite()

    # Добавляем тестовые классы
    suite.addTests(loader.loadTestsFromTestCase(TestLanguageDataProcessor))
    suite.addTests(loader.loadTestsFromTestCase(TestDataClasses))
    suite.addTests(loader.loadTestsFromTestCase(TestHelperMethods))

    # Запускаем тесты с подробным выводом
    runner = unittest.TextTestRunner(verbosity=2)
    return runner.run(suite)


if __name__ == '__main__':
    # Запуск тестов при прямом вызове файла
    run_all_tests()
```

run_test.py

```
"""
Демонстрационный файл для запуска модульных тестов
"""
import unittest
from test_main import run_all_tests


def main():
    """Основная функция запуска тестов"""
    print("=" * 70)
    print("РУБЕЖНЫЙ КОНТРОЛЬ №2 - МОДУЛЬНОЕ ТЕСТИРОВАНИЕ")
    print("Предметная область: Синтаксическая конструкция - Язык
программирования")
    print("=" * 70)
    print()
```

```
    print("ЗАПУСК ТЕСТОВ...")
    print()

    # Запуск всех тестов
    test_result = run_all_tests()

    print()
    print("=" * 70)
    print("ИТОГИ ТЕСТИРОВАНИЯ:")
    print("-" * 70)
    print(f"Всего тестов выполнено: {test_result.testsRun}")

    if test_result.failures:
        print(f"ПРОВАЛЕНО тестов: {len(test_result.failures)}")
        for test, traceback in test_result.failures:
            print(f"  - {test}")
    else:
        print("ПРОВАЛЕНО тестов: 0")

    if test_result.errors:
        print(f"ОШИБОК при выполнении: {len(test_result.errors)}")
        for test, traceback in test_result.errors:
            print(f"  - {test}")
    else:
        print("ОШИБОК при выполнении: 0")

    success_count = test_result.testsRun - len(test_result.failures) -
len(test_result.errors)
    success_rate = (success_count / test_result.testsRun * 100) if
test_result.testsRun > 0 else 0

    print(f"УСПЕШНО пройдено: {success_count} ({success_rate:.1f}%)")
    print("=" * 70)


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

Результат выполнения

```
test_many_to_many_relationship_integrity (test_main.TestLanguageDataProcessor.test_many_to_many_relationship_integrity)
Тест целостности связи многие-ко-многим ... ok
test_one_to_many_relationship_integrity (test_main.TestLanguageDataProcessor.test_one_to_many_relationship_integrity)
Тест целостности связи один-ко-многим ... ok
test_task_a1_structure_and_sorting (test_main.TestLanguageDataProcessor.test_task_a1_structure_and_sorting)
Тест структуры и сортировки задания A1 ... ok
test_task_a2_calculation_and_sorting (test_main.TestLanguageDataProcessor.test_task_a2_calculation_and_sorting)
Тест расчетов и сортировки задания A2 ... ok
test_task_a3_filtering (test_main.TestLanguageDataProcessor.test_task_a3_filtering)
Тест фильтрации задания A3 ... ok
test_construct_language_creation (test_main.TestDataClasses.test_construct_language_creation)
Тест создания объекта ConstructLanguage ... ok
test_programming_language_creation (test_main.TestDataClasses.test_programming_language_creation)
Тест создания объекта ProgrammingLanguage ... ok
test_syntax_construct_creation (test_main.TestDataClasses.test_syntax_construct_creation)
Тест создания объекта SyntaxConstruct ... ok
test_get_construct_by_name (test_main.TestHelperMethods.test_get_construct_by_name)
Тест метода get_construct_by_name ... ok
test_get_language_by_name (test_main.TestHelperMethods.test_get_language_by_name)
Тест метода get_language_by_name ... ok

----------------------------------------------------------------------
Ran 10 tests in 0.001s

OK

======================================================================
ИТОГИ ТЕСТИРОВАНИЯ:
----------------------------------------------------------------------
Всего тестов выполнено: 10
ПРОВАЛЕНО тестов: 0
ОШИБОК при выполнении: 0
УСПЕШНО пройдено: 10 (100.0%)
======================================================================
```