Защищено:
Гапанюк Ю.Е.

"__"_____2025 г.

Демонстрация:
Гапанюк Ю.Е.

"__"_____2025 г.

Отчет по лабораторной работе № 4 по курсу Парадигмы и конструкции языков программирования ГУИМЦ

Тема работы: "Шаблоны проектирования и модульное тестирование в Python."

13
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5Ц-54Б _____
(подпись)

Щетинин Д.С.

"__"_____2025 г.

Москва, МГТУ - 2025 г.

1. Тема и задание для выполнения лабораторной работы.

Тема работы: "Изучение шаблонов проектирования и модульного тестирования в Python."
Задание:

1. Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог. Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.
2. В модульных тестах необходимо применить следующие технологии:
    - TDD - фреймворк.
    - BDD - фреймворк.
    - Создание Mock-объектов.

TDD – (python -m unittest discover -s tests)
BDD – (python -m behave features/patterns.feature)
Mock-объектов – (python -m unittest tests/test_mocks.py)

После того, как все коды готовы, то необходимо запускать Terminal (Alt + F12), затем вводим все команды

pip install behave
python -m unittest discover -s tests)
python -m behave features/patterns.feature
python -m unittest tests/test_mocks.py

2. Листинг программы

Файл: run_behave.py

```
from behave.__main__ import main as behave_main
behave_main(["features/patterns.feature"])
```

Файл: features/steps/patterns_steps.py

```
from behave import *
from lab_python_fp import (
    VacancyFactory,
    ProgrammerVacancy,
    SalaryAdapter,
    LegacySalarySystem,
    VacancyProcessor,
    PositionFilterStrategy
)

@given('Фабрика вакансий')
def step_impl(context):
    context.factory = VacancyFactory

@when('Создаем вакансию программиста')
def step_impl(context):
    context.vacancy = context.factory.create_vacancy("programmer")

@then('Получаем вакансию с описанием "Программист Python"')
def step_impl(context):
    assert isinstance(context.vacancy, ProgrammerVacancy)
```

```
    assert context.vacancy.get_description() == "Программист Python"

@given('Адаптер системы зарплат')
def step_impl(context):
    context.adapter = SalaryAdapter(LegacySalarySystem())

@when('Рассчитываем зарплату для "Программист Python"')
def step_impl(context):
    context.salary = context.adapter.calculate_salary("Программист Python")

@then('Получаем зарплату 120000')
def step_impl(context):
    assert context.salary == 120000

@given('Процессор вакансий с фильтром программистов')
def step_impl(context):
    context.processor = VacancyProcessor(PositionFilterStrategy("Программист"))

@when('Фильтруем список вакансий')
def step_impl(context):
    context.vacancies = [
        {"position": "Программист Python"},
        {"position": "Аналитик"},
        {"position": "Программист Java"}
    ]
    context.filtered = context.processor.process(context.vacancies)

@then('Получаем только вакансии программистов')
def step_impl(context):
    assert len(context.filtered) == 2
    assert all("Программист" in v["position"] for v in context.filtered)
```

Файл: features/steps/process_steps.py

```
from behave import *
from lab_python_fp.process_data import DataProcessor

@given('имеются данные о вакансиях')
def step_impl(context):
    context.processor = DataProcessor()
    context.data = [
        {'job-name': 'Программист Python'},
        {'job-name': 'Аналитик'},
        {'job-name': 'Программист Java'}
    ]

@when('применяется конвейер обработки')
def step_impl(context):
    context.result = context.processor.process_pipeline(context.data)

@then('получаем только вакансии программистов с Python и зарплатой')
def step_impl(context):
    assert len(context.result) == 2
    assert all('Программист' in job for job in context.result)
    assert all('с опытом Python' in job for job in context.result)
    assert all('зарплата' in job for job in context.result)
```

Файл: features/patterns.feature

```
Feature: Проверка шаблонов проектирования

  Scenario: Создание вакансии через фабрику
    Given Фабрика вакансий
    When Создаем вакансию программиста
    Then Получаем вакансию с описанием "Программист Python"

  Scenario: Расчет зарплаты через адаптер
    Given Адаптер системы зарплат
    When Рассчитываем зарплату для "Программист Python"
    Then Получаем зарплату 120000

  Scenario: Фильтрация вакансий стратегией
    Given Процессор вакансий с фильтром программистов
    When Фильтруем список вакансий
    Then Получаем только вакансии программистов
```

Файл: features/process.feature

```
Feature: Обработка данных о вакансиях
  Проверка корректности обработки вакансий

  Scenario: Фильтрация вакансий программистов
    Given имеются данные о вакансиях
    When применяется конвейер обработки
    Then получаем только вакансии программистов с Python и зарплатой
```

Файл: lab_python_fp/adapter.py

```
class LegacySalarySystem:
    """Старая система расчета зарплат"""
    def get_salary(self, position_code):
        salaries = {
            "DEV": 120000,
            "ANL": 90000,
            "MNG": 150000
        }
        return salaries.get(position_code, 0)

class SalaryAdapter:
    """Адаптер для современной системы зарплат"""
    def __init__(self, legacy_system):
        self.legacy_system = legacy_system

    def calculate_salary(self, position_name):
        """Новый интерфейс для работы с вакансиями"""
        code = self._get_position_code(position_name)
        return self.legacy_system.get_salary(code)

    def _get_position_code(self, position_name):
        codes = {
            "Программист": "DEV",
            "Аналитик": "ANL",
            "Менеджер": "MNG"
        }
        for key in codes:
            if key in position_name:
                return codes[key]
        return "UNK"
```

Файл: lab_python_fp/cm_timer.py

```
import time
from contextlib import contextmanager


class cm_timer_1:
    def __enter__(self):
        self.start_time = time.time()

    def __exit__(self, exc_type, exc_val, exc_tb):
        print(f"time: {time.time() - self.start_time}")


@contextmanager
def cm_timer_2():
    start_time = time.time()
    yield
    print(f"time: {time.time() - start_time}")


if __name__ == '__main__':
    print("Тестирование cm_timer_1:")
    with cm_timer_1():
        time.sleep(1.5)

    print("\nТестирование cm_timer_2:")
    with cm_timer_2():
        time.sleep(1.5)
```

Файл: lab_python_fp/data_light.json

```
[
  {
    "job-name": "Программист Python",
    "salary": 120000
  },
  {
    "job-name": "Программист Java",
    "salary": 110000
  },
  {
    "job-name": "Аналитик данных",
    "salary": 90000
  },
  {
    "job-name": "Программист C++",
    "salary": 130000
  },
  {
    "job-name": "Веб-разработчик",
    "salary": 100000
  }
]
```

Файл: lab_python_fp/factory_method.py

```
from abc import ABC, abstractmethod
```

```
class Vacancy(ABC):
    """Абстрактный класс вакансии"""
    @abstractmethod
    def get_description(self):
        pass

class ProgrammerVacancy(Vacancy):
    """Конкретная вакансия программиста"""
    def get_description(self):
        return "Программист Python"

class AnalystVacancy(Vacancy):
    """Конкретная вакансия аналитика"""
    def get_description(self):
        return "Аналитик данных"

class VacancyFactory:
    """Фабрика создания вакансий"""
    @staticmethod
    def create_vacancy(vacancy_type):
        if vacancy_type == "programmer":
            return ProgrammerVacancy()
        elif vacancy_type == "analyst":
            return AnalystVacancy()
        raise ValueError("Неизвестный тип вакансии")
```

Файл: lab_python_fp/field.py

```
def field(items, *args):
    assert len(args) > 0
    for item in items:
        if len(args) == 1:
            value = item.get(args[0])
            if value is not None:
                yield value
        else:
            result = {}
            has_values = False
            for key in args:
                value = item.get(key)
                if value is not None:
                    result[key] = value
                    has_values = True
            if has_values:
                yield result



if __name__ == "__main__":
    goods = [
        {'title': 'Ковер', 'price': 2000, 'color': 'green'},
        {'title': 'Диван для отдыха', 'color': 'black'}
    ]

    # Тест 1
    print("Test 1:")
    for item in field(goods, 'title'):
        print(item)
```

```
    # Тест 2
    print("\nTest 2:")
    for item in field(goods, 'title', 'price'):
        print(item)
```

Файл: lab_python_fp/gen_random.py

```
import random

def gen_random(num_count, begin, end):
    for _ in range(num_count):
        yield random.randint(begin, end)



if __name__ == "__main__":
    # Тест
    print("Test:")
    for num in gen_random(5, 1, 3):
        print(num, end=" ")
    print()
```

Файл: lab_python_fp/print_result.py

```
def print_result(func):
    def wrapper(*args, **kwargs):
        result = func(*args, **kwargs)
        print(func.__name__)
        if isinstance(result, list):
            for item in result:
                print(item)
        elif isinstance(result, dict):
            for key, value in result.items():
                print(f"{key} = {value}")
        else:
            print(result)
        return result
    return wrapper

@print_result
def test_1():
    return 1

@print_result
def test_2():
    return 'iu5'

@print_result
def test_3():
    return {'a': 1, 'b': 2}

@print_result
def test_4():
    return [1, 2]

if __name__ == '__main__':
    print('!!!!!!!!')
```

```
    test_1()
    test_2()
    test_3()
    test_4()
```

Файл: lab_python_fp/process_data.py

```
import json
import sys
from .gen_random import gen_random
from .unique import Unique
from .print_result import print_result
from .cm_timer import cm_timer_1


class DataProcessor:
    """Класс для обработки данных о вакансиях"""

    @staticmethod
    def load_data(path):
        """Загрузка данных из JSON файла"""
        with open(path, 'r', encoding='utf-8') as f:
            return json.load(f)

    @print_result
    def f1(self, data):
        """Этап 1: Уникальные названия с сортировкой"""
        job_names = [item['job-name'] for item in data]
        return sorted(Unique(job_names, ignore_case=True), key=str.lower)

    @print_result
    def f2(self, job_names):
        """Этап 2: Фильтрация программистов"""
        return [name for name in job_names if name.lower().startswith('программист')]

    @print_result
    def f3(self, job_names):
        """Этап 3: Добавление Python-опыта"""
        return [f"{name} с опытом Python" for name in job_names]

    @print_result
    def f4(self, job_names):
        """Этап 4: Добавление зарплат"""
        salaries = list(gen_random(len(job_names), 100000, 200000))
        return [f"{job}, зарплата {salary} руб." for job, salary in zip(job_names, salaries)]

    def process_pipeline(self, data):
        """Запуск полного конвейера обработки"""
        return self.f4(self.f3(self.f2(self.f1(data))))


if __name__ == '__main__':
    if len(sys.argv) < 2:
        print("Usage: python process_data.py <path_to_json>")
        sys.exit(1)

    processor = DataProcessor()
    data = processor.load_data(sys.argv[1])
    with cm_timer_1():
        result = processor.process_pipeline(data)
```

Файл: lab_python_fp/sort.py

```
data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]

if __name__ == '__main__':
    # Без lambda
    result = sorted(data, key=abs, reverse=True)
    print("Без использования lambda-функции:", result)

    # С lambda
    result_with_lambda = sorted(data, key=lambda x: abs(x), reverse=True)
    print("С использованием lambda-функции:", result_with_lambda)
```

Файл: lab_python_fp/strategy.py

```
from abc import ABC, abstractmethod


class FilterStrategy(ABC):
    """Абстрактная стратегия фильтрации"""

    @abstractmethod
    def filter(self, vacancies):
        pass


class PositionFilterStrategy(FilterStrategy):
    """Конкретная стратегия фильтрации по должности"""

    def __init__(self, position_keyword):
        self.position_keyword = position_keyword

    def filter(self, vacancies):
        return [v for v in vacancies if self.position_keyword in v["position"]]


class VacancyProcessor:
    """Контекст, использующий стратегию"""

    def __init__(self, strategy):
        self.strategy = strategy

    def process(self, vacancies):
        return self.strategy.filter(vacancies)
```

Файл: lab_python_fp/unique.py

```
class Unique(object):
    def __init__(self, items, **kwargs):
        self.ignore_case = kwargs.get('ignore_case', False)
        self.items = iter(items)
        self.seen = set()

    def __next__(self):
        while True:
            item = next(self.items)
            key = item.lower() if self.ignore_case and isinstance(item, str) else item
```

```
            if key not in self.seen:
                self.seen.add(key)
                return item

    def __iter__(self):
        return self


if __name__ == "__main__":
    # Тест 1
    print("Test 1 (numbers):")
    data = [1, 1, 1, 1, 1, 2, 2, 2, 2, 2]
    for item in Unique(data):
        print(item, end=" ")
    print()

    # Тест 2
    print("\nTest 2 (strings):")
    data = ['a', 'A', 'b', 'B', 'a', 'A', 'b', 'B']
    for item in Unique(data, ignore_case=True):
        print(item, end=" ")
    print()
```

Файл: tests/test_mocks.py

```
from unittest import TestCase, mock
from lab_python_fp.process_data import DataProcessor


class TestWithMocks(TestCase):
    @mock.patch('lab_python_fp.process_data.gen_random')
    def test_f4_with_mocked_salaries(self, mock_gen):
        mock_gen.return_value = [150000, 180000]
        processor = DataProcessor()
        result = processor.f4(['Job1', 'Job2'])

        self.assertEqual(result, [
            'Job1, зарплата 150000 руб.',
            'Job2, зарплата 180000 руб.'
        ])
        mock_gen.assert_called_once_with(2, 100000, 200000)
```

Файл: tests/test_patterns.py

```
import unittest
from unittest.mock import Mock, patch
from lab_python_fp.factory_method import VacancyFactory, ProgrammerVacancy
from lab_python_fp.adapter import LegacySalarySystem, SalaryAdapter
from lab_python_fp.strategy import VacancyProcessor, PositionFilterStrategy

class TestFactoryMethod(unittest.TestCase):
    def test_create_programmer_vacancy(self):
        vacancy = VacancyFactory.create_vacancy("programmer")
        self.assertIsInstance(vacancy, ProgrammerVacancy)

class TestAdapterPattern(unittest.TestCase):
    def test_salary_calculation(self):
        legacy_system = LegacySalarySystem()
        adapter = SalaryAdapter(legacy_system)
```

```
        self.assertEqual(adapter.calculate_salary("Программист Python"), 120000)

class TestStrategyPattern(unittest.TestCase):
    def test_programmer_filter(self):
        processor = VacancyProcessor(PositionFilterStrategy("Программист"))
        vacancies = [
            {"position": "Программист Python"},
            {"position": "Аналитик"},
            {"position": "Программист Java"}
        ]
        filtered = processor.process(vacancies)
        self.assertEqual(len(filtered), 2)

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

Файл: tests/test_process.py

```
import unittest
from unittest.mock import patch
from lab_python_fp.process_data import DataProcessor

class TestDataProcessor(unittest.TestCase):
    def setUp(self):
        self.processor = DataProcessor()
        self.test_data = [
            {'job-name': 'Программист Python'},
            {'job-name': 'Аналитик'},
            {'job-name': 'Программист Java'}
        ]

    def test_f1_sorts_and_returns_unique_names(self):
        result = self.processor.f1(self.test_data)
        self.assertEqual(result, ['Аналитик', 'Программист Java', 'Программист Python'])

    def test_f2_filters_only_programmers(self):
        names = ['Программист Python', 'Аналитик', 'Программист Java']
        result = self.processor.f2(names)
        self.assertEqual(result, ['Программист Python', 'Программист Java'])

    @patch('lab_python_fp.process_data.gen_random')
    def test_f4_with_mocked_salaries(self, mock_gen):
        mock_gen.return_value = [150000, 180000]
        result = self.processor.f4(['Job1', 'Job2'])
        self.assertEqual(result, [
            'Job1, зарплата 150000 руб.',
            'Job2, зарплата 180000 руб.'
        ])
        mock_gen.assert_called_once_with(2, 100000, 200000)

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

3. Результаты работы программы

```
PS C:\Users\Денис\Desktop\Python-4lab> pip install behave
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: behave in c:\users\денис\appdata\roaming\python\python313\site-packages (1.3.3)
Requirement already satisfied: cucumber-tag-expressions>=4.1.0 in
c:\users\денис\appdata\roaming\python\python313\site-packages (from behave) (8.0.0)
Requirement already satisfied: cucumber-expressions>=17.1.0 in
c:\users\денис\appdata\roaming\python\python313\site-packages (from behave) (18.0.1)
Requirement already satisfied: parse>=1.18.0 in c:\users\денис\appdata\roaming\python\python313\site-packages
(from behave) (1.20.2)
Requirement already satisfied: parse-type>=0.6.0 in c:\users\денис\appdata\roaming\python\python313\site-
packages (from behave) (0.6.6)
Requirement already satisfied: six>=1.15.0 in c:\users\денис\appdata\roaming\python\python313\site-packages
(from behave) (1.17.0)
Requirement already satisfied: colorama>=0.3.7 in c:\users\денис\appdata\roaming\python\python313\site-packages
(from behave) (0.4.6)

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip

PS C:\Users\Денис\Desktop\Python-4lab> python -m unittest discover -s tests
f4
Job1, зарплата 150000 руб.
Job2, зарплата 180000 руб.
....f1
Аналитик
Программист Java
Программист Python
.f2
Программист Python
Программист Java
.f4
Job1, зарплата 150000 руб.
Job2, зарплата 180000 руб.
.
----------------------------------------------------------------------
Ran 7 tests in 0.007s

OK
```

```
PS C:\Users\Денис\Desktop\Python-4lab> python -m behave features/patterns.feature
USING RUNNER: behave.runner:Runner
Feature: Проверка шаблонов проектирования # features/patterns.feature:1

  Then Получаем вакансию с описанием "Программист Python" # features/steps/patterns_steps.py:19 0.001s
  Then Получаем вакансию с описанием "Программист Python" # features/steps/patterns_steps.py:19
  Then Получаем зарплату 120000                # features/steps/patterns_steps.py:32 0.001s
  Then Получаем зарплату 120000                # features/steps/patterns_steps.py:32
  Then Получаем только вакансии программистов  # features/steps/patterns_steps.py:49 0.001s
  Then Получаем только вакансии программистов  # features/steps/patterns_steps.py:49
1 feature passed, 0 failed, 0 skipped
3 scenarios passed, 0 failed, 0 skipped
9 steps passed, 0 failed, 0 skipped
Took 0min 0.009s
```

```
PS C:\Users\Денис\Desktop\Python-4lab> python -m unittest tests/test_mocks.py
f4
Job1, зарплата 150000 руб.
Job2, зарплата 180000 руб.
.
----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
PS C:\Users\Денис\Desktop\Python-4lab>
```