

ОТЧЕТ

По лр-4

Дисциплина «Парадигмы и конструкции языков программирования»

Студент: Коваленко Е.

Группа: ИБМ3-34Б

В качестве предметной области была выбрана университетская среда: преподаватели и студенты

1. Factory Method (TDD-фреймворк)

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

```
class UniversityMember(ABC):
```

```
    @abstractmethod
```

```
    def get_role(self):
```

```
        pass
```

```
class Professor(UniversityMember):
```

```
    def get_role(self):
```

```
        return "Professor"
```

```
class Student(UniversityMember):
```

```
    def get_role(self):
```

```
        return "Student"
```

```
class MemberFactory(ABC):
```

```
    @abstractmethod
```

```
    def create_member(self):
```

```
        pass
```

```
class ProfessorFactory(MemberFactory):
```

```
    def create_member(self):
```

```
        return Professor()
```

```
class StudentFactory(MemberFactory):
```

```
    def create_member(self):
```

```
        return Student()
```

```

def client_code(factory: MemberFactory):
    member = factory.create_member()
    print(f"Created a university member with role: {member.get_role()}")

class TestMemberFactory(unittest.TestCase):
    def test_professor_factory(self):
        factory = ProfessorFactory()
        prof = factory.create_member()
        self.assertEqual(prof.get_role(), "Professor")

    def test_student_factory(self):
        factory = StudentFactory()
        student = factory.create_member()
        self.assertEqual(student.get_role(), "Student")

if __name__ == "__main__":
    client_code(ProfessorFactory())
    client_code(StudentFactory())
    print("\nRunning tests...\n")
    unittest.main(argv=['first-arg-is-ignored'], exit=False)

```

The screenshot shows two windows side-by-side. The left window is titled 'Factory.py - C:\Users\User\Desktop\Factory.py (3.13.1)' and contains the Python code from the previous block. The right window is titled 'IDLE Shell 3.13.1' and shows the execution of the code. The shell output includes the imports, class definitions, and the execution of the unittest.main() command. It also shows the creation of university members and the running of tests, which pass successfully.

```

File Edit Format Run Options Window Help
File Edit Shell Debug Options Window Help
Python 3.13.1 (tags/v3.13.1:0671451, Dec 3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

=====
RESTART: C:\Users\User\Desktop\Factory.py =====
Created a university member with role: Professor
Created a university member with role: Student

Running tests...

...
-----
Ran 2 tests in 0.005s
OK

```

2. Chain of Responsibility + Mock-объекты

from abc import ABC, abstractmethod

```
import unittest

from unittest.mock import Mock


class HelpRequest:

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


class Handler(ABC):

    def __init__(self):
        self._next_handler = None

    def set_next(self, handler):
        self._next_handler = handler
        return handler

    @abstractmethod
    def handle(self, request):
        pass


class Assistant(Handler):

    def handle(self, request):
        if "простой вопрос" in request.description:
            return "Assistant решает вопрос."
        elif self._next_handler:
            return self._next_handler.handle(request)
        else:
            return "Запрос не может быть обработан."


class Professor(Handler):

    def handle(self, request):
```

```
if request.description.strip() == "сложный вопрос" or
request.description.strip().startswith("сложный вопрос "):

    return "Professor решает вопрос."

elif self._next_handler:

    return self._next_handler.handle(request)

else:

    return "Запрос не может быть обработан."


class Dean(Handler):

    def handle(self, request):

        return "Dean решает вопрос."


def demo():

    assistant = Assistant()

    professor = Professor()

    dean = Dean()

    assistant.set_next(professor).set_next(dean)

    requests = [
        HelpRequest("простой вопрос про расписание"),
        HelpRequest("сложный вопрос"),
        HelpRequest("очень сложный вопрос по политике факультета")
    ]

    for r in requests:

        print(assistant.handle(r))

class TestChainWithMock(unittest.TestCase):

    def test_handle_calls_next_when_not_handled(self):

        assistant = Assistant()
```

```

mock_handler = Mock()
mock_handler.handle.return_value = "Mocked handler processed"
assistant.set_next(mock_handler)

req = HelpRequest("сложный вопрос")

response = assistant.handle(req)

mock_handler.handle.assert_called_once_with(req)
self.assertEqual(response, "Mocked handler processed")

def test_assistant_handles_easy_question(self):
    assistant = Assistant()
    req = HelpRequest("простой вопрос")
    self.assertEqual(assistant.handle(req), "Assistant решает вопрос.")

if __name__ == "__main__":
    demo()
    print("\nЗапуск тестов...\n")
    unittest.main(argv=['first-arg-is-ignored'], exit=False)

```

The screenshot shows two windows side-by-side. The left window is titled 'Chain.py - C:\Users\User\Desktop\Chain.py (3.13.1)' and contains Python code for a linked list class and a test case. The right window is titled 'IDLE Shell 3.13.1' and shows the execution of this code in a terminal-like interface.

```

Chain.py - C:\Users\User\Desktop\Chain.py (3.13.1)
File Edit Format Run Options Window Help
def demo():
    assistant = Assistant()
    professor = Professor()
    dean = Dean()

    assistant.set_next(professor).set_next(dean)

    requests = [
        HelpRequest("простой вопрос про расписание"),
        HelpRequest("сложный вопрос"),
        HelpRequest("очень сложный вопрос по полит")
    ]

    for r in requests:
        print(assistant.handle(r))

class TestChainWithMock(unittest.TestCase):
    def test_handle_calls_next_when_not_handled(self):
        pass

```

```

IDLE Shell 3.13.1
File Edit Shell Debug Options Window Help
Python 3.13.1 (tags/v3.13.1:0671451, Dec 3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
=====
RESTART: C:\Users\User\Desktop\Chain.py =====
Assistant решает вопрос.
Professor решает вопрос.
Dean решает вопрос.

Запуск тестов...
..
Ran 2 tests in 0.005s
OK

```

3. Bridge + TDD

```

from abc import ABC, abstractmethod
import unittest

```

```
class Person(ABC):  
    def __init__(self, role_impl):  
        self._role_impl = role_impl
```

```
    @abstractmethod  
    def perform_duty(self):  
        pass
```

```
class RoleImpl(ABC):  
    @abstractmethod  
    def duty(self):  
        pass
```

```
class StudentRole(RoleImpl):  
    def duty(self):  
        return "Студент учится и выполняет домашние задания."
```

```
class TeacherRole(RoleImpl):  
    def duty(self):  
        return "Преподаватель читает лекции и проверяет работы."
```

```
class Student(Person):  
    def perform_duty(self):  
        return self._role_impl.duty()
```

```
class Teacher(Person):  
    def perform_duty(self):  
        return self._role_impl.duty()
```

```
class TestUniversityBridge(unittest.TestCase):

    def test_student_duty(self):
        student_role = StudentRole()
        student = Student(student_role)
        self.assertEqual(student.perform_duty(), "Студент учится и выполняет домашние задания.")

    def test_teacher_duty(self):
        teacher_role = TeacherRole()
        teacher = Teacher(teacher_role)
        self.assertEqual(teacher.perform_duty(), "Преподаватель читает лекции и проверяет работы.")

    def test_role_swap(self):
        student_role = StudentRole()
        teacher_role = TeacherRole()
        person = Student(student_role)
        self.assertEqual(person.perform_duty(), "Студент учится и выполняет домашние задания.")
        person._role_impl = teacher_role
        self.assertEqual(person.perform_duty(), "Преподаватель читает лекции и проверяет работы.")

if __name__ == "__main__":
    student = Student(StudentRole())
    teacher = Teacher(TeacherRole())

    print("Студент:", student.perform_duty())
    print("Преподаватель:", teacher.perform_duty())
    print()
```

```
print("Запуск тестов...\n")
```

```
unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

The screenshot shows two windows side-by-side. On the left is a code editor window titled "Bridge.py - C:\Users\User\Desktop\Bridge.py (3.13.1)". It contains Python code defining classes for Student and Teacher roles, and a TestUniversityBridge test class. On the right is an "IDLE Shell 3.13.1" window showing the output of running the tests. The terminal output includes the Python version, copyright information, a restart message, and descriptions of the Student and Teacher roles. It also shows the test run starting, running 3 tests in 0.010s, and concluding with an OK status.

```
Bridge.py - C:\Users\User\Desktop\Bridge.py (3.13.1)
File Edit Format Run Options Window Help
class Student(Person):
    def perform_duty(self):
        return self._role_impl.duty()
class Teacher(Person):
    def perform_duty(self):
        return self._role_impl.duty()
class TestUniversityBridge(unittest.TestCase):
    def test_student_duty(self):
        student_role = StudentRole()
        student = Student(student_role)
        self.assertEqual(student.perform_duty(), "Студент учится и выполняет домашние задания.")
    def test_teacher_duty(self):
        teacher_role = TeacherRole()
        teacher = Teacher(teacher_role)
        self.assertEqual(teacher.perform_duty(), "Преподаватель читает лекции и проверяет работы.")

IDLE Shell 3.13.1
File Edit Shell Debug Options Window Help
Python 3.13.1 (tags/v3.13.1:0671451, Dec  3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
=====
===== RESTART: C:\Users\User\Desktop\Bridge.py =====
Студент: Студент учится и выполняет домашние задания.
Преподаватель: Преподаватель читает лекции и проверяет работы.

Запуск тестов...
...
-----
Ran 3 tests in 0.010s
OK
```