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 лабораторной работе № 4
“Шаблоны проектирования и модульное тестирование в Python.”

Выполнил:
Студент группы ИУ5-36Б
Левочкин В.В.

Преподаватель:
Нардид. А.Н.

Москва 2025

Задание

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.

1.

bdd_tests.py

```
importpytest
frompytest_bdd importscenarios, given, when, then, parsers
fromLab_4.test_stateimportCar, OnRoadState, AtGasStationState, InSerivceState


scenarios('features/car_behavior.feature')


@pytest.fixture
defcar():
    returnCar(OnRoadState())


@pytest.fixture
defcontext():
    return{'output': ''}


@given("автомобиль находится на дороге")
defcar_on_road(car):
    car.change_state(OnRoadState())
    returncar


@given("автомобиль находится на заправке")
defcar_at_gas_station(car):
    car.change_state(AtGasStationState())
    returncar


@given("автомобиль находится в сервисе")
defcar_in_service(car):
    car.change_state(InSerivceState())
    returncar


@when("я пытаюсь вести автомобиль")
deftry_to_drive(car, context, capsys):
    car.drive()
    captured=capsys.readouterr()
    context['output'] =captured.out.strip()
```

```
@when("я пытаюсь припарковать автомобиль")
deftry_to_park(car, context, capsys):
    car.park()
    captured=capsys.readouterr()
    context['output'] =captured.out.strip()


@when("я пытаюсь заправить автомобиль")
deftry_to_refuel(car, context, capsys):
    car.refuel()
    captured=capsys.readouterr()
    context['output'] =captured.out.strip()


@when("я меняю состояние на заправку")
defchange_to_gas_station(car):
    car.change_state(AtGasStationState())


@when("я меняю состояние на сервис")
defchange_to_in_service(car):
    car.change_state(InSerivceState())


@then(parsers.parse("вывод должен содержать '{expected_text}'"))
defoutput_contains_expected_text(context, expected_text):
    assertexpected_textincontext['output'], f"Expected '{expected_text}' in
output, but got: {context['output']}"
```

Mock_tests.py

```
importpytest
fromunittest.mockimportMock, patch, MagicMock
fromLab_4.test_wrapperimport(
    LocalFileSystem,
    RemoteFileSystem,
    SourceLocalFileSystem,
    SourceRemoteFileSystem,
    FileServer,
    ISource
)


classTestLocalFileSystem:
    deftest_local_read(self):
        lfs=LocalFileSystem()
        result=lfs.local_read()
        assertresult=='читаю файл на локальном устройстве...'


    deftest_local_write(self):
        lfs=LocalFileSystem()
        result=lfs.local_write()
        assertresult=='записываю в файл на локальном устройстве...'


classTestRemoteFileSystem:
```

```
    deftest_remote_filesystem_initialization(self):
        rfs=RemoteFileSystem('127.0.0.1')
        assertrfs.host=='127.0.0.1'


    deftest_connect(self):
        rfs=RemoteFileSystem('127.0.0.1')
        result=rfs.connect()
        assertresult=='подключаюсь к удалённому устройству...'


    deftest_remote_read(self):
        rfs=RemoteFileSystem('127.0.0.1')
        result=rfs.remote_read()
        assertresult=='читаю файл на удалённом устройстве...'


    deftest_remote_write(self):
        rfs=RemoteFileSystem('127.0.0.1')
        result=rfs.remote_write()
        assertresult=='записываю в файл на удалённом устройстве...'


classTestSourceLocalFileSystem:
    deftest_source_local_read(self):
        source=SourceLocalFileSystem()
        result=source.read()
        assertresult=='читаю файл на локальном устройстве...'


    deftest_source_local_write(self):
        source=SourceLocalFileSystem()
        result=source.write()
        assertresult=='записываю в файл на локальном устройстве...'


classTestSourceRemoteFileSystem:
    deftest_source_remote_read(self):
        source=SourceRemoteFileSystem('127.0.0.1')
        result=source.read()
        expected='подключаюсь к удалённому устройству...\nчитаю файл на
удалённом устройстве...'
        assertresult==expected


    deftest_source_remote_write(self):
        source=SourceRemoteFileSystem('127.0.0.1')
        result=source.write()
        expected='подключаюсь к удалённому устройству...\nзаписываю в файл на
удалённом устройстве...'
        assertresult==expected


classTestFileServer:
    deftest_fileserver_initialization(self):
        mock_source=Mock(spec=ISource)
        server=FileServer(mock_source)
        assertserver.source==mock_source
```

```
    def test_fileserver_read_with_mock(self):
        mock_source=Mock(spec=ISource)
        mock_source.read.return_value ='mock read result'


        server=FileServer(mock_source)
        result=server.read()


        assert result=='mock read result'


    def test_fileserver_write_with_mock(self):
        mock_source=Mock(spec=ISource)
        mock_source.write.return_value ='mock write result'


        server=FileServer(mock_source)
        result=server.write()


        assert result=='mock write result'


    def test_fileserver_with_local_source(self):
        local_source=SourceLocalFileSystem()
        server=FileServer(local_source)


        read_result=server.read()
        write_result=server.write()


        assert read_result=='читаю файл на локальном устройстве...'
        assert write_result=='записываю в файл на локальном устройстве...'


    def test_fileserver_with_remote_source(self):
        remote_source=SourceRemoteFileSystem('127.0.0.1')
        server=FileServer(remote_source)


        read_result=server.read()
        write_result=server.write()


        expected_read='подключаюсь к удалённому устройству...\nчитаю файл на
удалённом устройстве...'
        expected_write='подключаюсь к удалённому устройству...\nзаписываю в файл
на удалённом устройстве...'


        assert read_result==expected_read
        assert write_result==expected_write


    @patch('wrapper.SourceLocalFileSystem')
```

```
deftest_fileserver_with_patched_local_source(self, mock_local_class):
    mock_source=Mock()
    mock_source.read.return_value ='patched local read'
    mock_source.write.return_value ='patched local write'
    mock_local_class.return_value =mock_source


    server=FileServer(mock_local_class())


    assertserver.read() =='patched local read'
    assertserver.write() =='patched local write'


@patch('wrapper.SourceRemoteFileSystem')
deftest_fileserver_with_patched_remote_source(self, mock_remote_class):
    mock_source=Mock()
    mock_source.read.return_value ='patched remote read'
    mock_source.write.return_value ='patched remote write'
    mock_remote_class.return_value =mock_source


    server=FileServer(mock_remote_class('127.0.0.1'))


    assertserver.read() =='patched remote read'
    assertserver.write() =='patched remote write'


deftest_fileserver_dependency_injection(self):
    mock_source=MagicMock()
    mock_source.read.return_value ="читаю из mock источника"
    mock_source.write.return_value ="пишу в mock источник"


    server=FileServer(mock_source)


    assertserver.read() =="читаю из mock источника"
    assertserver.write() =="пишу в mock источник"


    mock_source.read.assert_called_once()
    mock_source.write.assert_called_once()


deftest_fileserver_with_different_mock_behaviors(self):
    """Тестирование FileServer с разным поведением mock объектов"""
    # Mock с разными возвращаемыми значениями для последовательных вызовов
    mock_source=Mock()
    mock_source.read.side_effect =['first read', 'second read']
    mock_source.write.side_effect =['first write', 'second write']


    server=FileServer(mock_source)


    assertserver.read() =='first read'
    assertserver.write() =='first write'
```

```
        assertserver.read() =='second read'
        assertserver.write() =='second write'


        assertmock_source.read.call_count ==2
        assertmock_source.write.call_count ==2


if__name__=='__main__':
    pytest.main([__file__, '-v'])
```

Test_abstract_factory.py

```
fromabcimportABCMeta, abstractmethod


classCar(metaclass=ABCMeta):
    @abstractmethod
    defdrive(self):
        pass


classWolkswagen(Car):
    defdrive(self):
        print('Wolkswagen едет на бензине')


classTesla(Car):
    defdrive(self):
        print('Tesla едет на электричестве')


classCarDealer(metaclass=ABCMeta):
    @abstractmethod
    defbuyCar(self):
        pass


classPetrolCarDealer(CarDealer):
    defbuyCar(self):
        returnWolkswagen()


classElectricCarDealer(CarDealer):
    defbuyCar(self):
        returnTesla()


if__name__=='__main__':
    pcd=PetrolCarDealer()
    ecd=ElectricCarDealer()
    pcd.buyCar().drive()
    ecd.buyCar().drive()
```

Test_state.py

```
fromabcimportABCMeta, abstractmethod
```

```
classState(metaclass=ABCMeta):
    @abstractmethod
    defdrive(self):
        pass


    @abstractmethod
    defpark(self):
        pass


    @abstractmethod
    defrefuel(self):
        pass


classInSerivceState(State):
    defdrive(self):
        return'не может ездить в сервисе!'


    defpark(self):
        return'припарковалась в сервис'


    defrefuel(self):
        return'не может заправляться в сервисе!'


classOnRoadState(State):
    defdrive(self):
        return'едет по дороге'


    defpark(self):
        return'припарковалась'


    defrefuel(self):
        return'не может заправляться в дороге!'


classAtGasStationState(State):
    defdrive(self):
        return'едет по заправке'


    defpark(self):
        return'припарковалась на заправке'


    defrefuel(self):
        return'заправляется...'



classCar:
    def__init__(self, state):
        self._state=state
```

```
    defchange_state(self, state):
        self._state=state


    defdrive(self):
        self._execute('drive')


    defpark(self):
        self._execute('park')


    defrefuel(self):
        self._execute('refuel')


    def_execute(self, operation: str) -> None:
        func=getattr(self._state, operation)
        print('Машина {}.'.format(func()))


if__name__=='__main__':
    car=Car(OnRoadState())
    car.drive()
    car.change_state(AtGasStationState())
    car.park()
    car.refuel()
    car.change_state(OnRoadState())
    car.drive()
    car.change_state(InSerivceState())
    car.park()
```

Test_tdd.py

```
importpytest


fromtest_abstract_factoryimport(
    Car,
    Wolkswagen,
    Tesla,
    CarDealer,
    PetrolCarDealer,
    ElectricCarDealer
)


# Остальной код остается без изменений...
classTestWolkswagen:
    deftest_wolkswagen_drive_method(self, capsys):
        car=Wolkswagen()
        car.drive()
        captured=capsys.readouterr()
        assertcaptured.out.strip() =="Wolkswagen едет на бензине"
```

```
classTestTesla:
    deftest_tesla_drive_method(self, capsys):
        car=Tesla()
        car.drive()
        captured=capsys.readouterr()
        assertcaptured.out.strip() =="Tesla едет на электричестве"


classTestPetrolCarDealer:
    deftest_petrol_car_dealer_buy_car_returns_wolkswagen(self):
        dealer=PetrolCarDealer()
        car=dealer.buyCar()
        assertisinstance(car, Wolkswagen)
        assertisinstance(car, Car)


    deftest_petrol_car_dealer_buy_car_returns_new_instance_each_time(self):
        dealer=PetrolCarDealer()
        car1=dealer.buyCar()
        car2=dealer.buyCar()
        assertcar1isnotcar2
        assertisinstance(car1, Wolkswagen)
        assertisinstance(car2, Wolkswagen)


classTestElectricCarDealer:
    deftest_electric_car_dealer_buy_car_returns_tesla(self):
        dealer=ElectricCarDealer()
        car=dealer.buyCar()
        assertisinstance(car, Tesla)
        assertisinstance(car, Car)


    deftest_electric_car_dealer_buy_car_returns_new_instance_each_time(self):
        dealer=ElectricCarDealer()
        car1=dealer.buyCar()
        car2=dealer.buyCar()
        assertcar1isnotcar2
        assertisinstance(car1, Tesla)
        assertisinstance(car2, Tesla)


if__name__=='__main__':
    pytest.main([__file__, '-v'])
```

Test_wrapper.py

```
classLocalFileSystem:
    deflocal_read(self):
        return'читаю файл на локальном устройстве...'


    deflocal_write(self):
        return'записываю в файл на локальном устройстве...'


classRemoteFileSystem:
    def__init__(self, host):
        self.host=host
```

```
    defconnect(self):
        return'подключаюсь к удалённому устройству...'


    defremote_read(self):
        return'читаю файл на удалённом устройстве...'


    defremote_write(self):
        return'записываю в файл на удалённом устройстве...'


classISource:
    defread(self):
        raiseNotImplementedError()


    defwrite(self):
        raiseNotImplementedError()


classSourceLocalFileSystem(ISource, LocalFileSystem):
    defread(self):
        returnself.local_read()


    defwrite(self):
        returnself.local_write()


classSourceRemoteFileSystem(ISource, RemoteFileSystem):
    defread(self):
        returnf'{self.connect()}\n{self.remote_read()}'


    defwrite(self):
        returnf'{self.connect()}\n{self.remote_write()}'



classFileServer:
    def__init__(self, source: ISource):
        self.source=source


    defread(self):
        returnself.source.read()


    defwrite(self):
        returnself.source.write()



if__name__=='__main__':
    lfs=FileServer(SourceLocalFileSystem())
    rfs=FileServer(SourceRemoteFileSystem('127.0.0.1'))
```

```
print(lfs.read())
print(rfs.write())
```

Вывод программы