Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления (ИУ5)

О т ч е т

по лабораторной работе №4

Дисциплина: Разработка Интернет-Приложений

Студент гр. ИУ5-53Б _____ Назаров М.М.

(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Гапанюк Ю.Е.

(Подпись, дата) (И.О. Фамилия)

Москва, 2020

1. Цель работы

Цель лабораторной работы: изучение реализации шаблонов проектирования и возможностей модульного тестирования в языке Python.

2. Задание

1. Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог.
2. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:

- TDD - фреймворк.
- BDD - фреймворк.
- Создание Mock-объектов.

3. Код программы

factorymethod.py

```
from __future__ import annotations
from abc import ABC, abstractmethod

"пораждающий паттерн"


class Deliver(ABC):
    @abstractmethod
    def devlivery_method(self):
        pass

    def some_operation(self) -> str:
        # Вызываем фабричный метод, чтобы получить объект-продукт.
        delivery = self.devlivery_method()
        if type(delivery) == str:
            return f"DELIVER: {delivery}"
        else:
            result = f"DELIVER: The product {delivery.operation()}"
        return result


class PlaneDelivery(Deliver):
    dictofweather = {"Очень плохая погода": 0, "Плохая погода": 1, "средняя": 2, "нор
мальная": 3, "Хорошая": 4}

    def __init__(self):
        self._fuel = 0
        self._weather = None

    def prepareflight(self, fueltoadd, weather):
        self.refueling(fueltoadd)
        self.updateweather(weather)

    def updateweather(self, weather):
```

```
        if weather not in self.dictofweather.keys():
            print("Pls give right weather status")
        else:
            self._weather = weather

    def refueling(self, count):
        self._fuel += count

    def isfuel(self):
        if self._fuel >= 10:
            self._fuel -= 10
            return True
        else:
            return False

    def checkweather(self):
        if self._weather == None:
            return 0
        if self.dictofweather[self._weather] >= 2:
            return 1
        else:
            return -1

    def devlivery_method(self):
        if self.isfuel():
            if self.checkweather() > 0:
                return Coal()
            elif self.checkweather() == -1:
                return "Sorry, can't deliver your product due to weahter."
            else:
                return "Pleas contact to dispatcher and update weather status"
        else:
            return "delivering can't be done no fuel."


class CarDelivery(Deliver):
    def __init__(self):
        self._fuel = 0

    def isfuel(self):
        if self._fuel >= 5:
            self._fuel -= 5
            return True
        else:
            return False

    def refueling(self, count):
        self._fuel += count

    def devlivery_method(self):
        if self.isfuel():
```

```
            return Wood()
        else:
            return "delivering can't be done no fuel."


class Product(ABC):

    @abstractmethod
    def operation(self) -> str:
        pass


class Coal(Product):
    def operation(self) -> str:
        return "Coal are delivered"


class Wood(Product):
    def operation(self) -> str:
        return "Wood are delivered"


def client_code(deliver: Deliver) -> None:
    print(f"Client: I'm don't now who is deliver exactly.\n"
          f"{deliver.some_operation()}")


if __name__ == "__main__":
    Plane = PlaneDelivery()
    car = CarDelivery()
    print("Launched with the car.")
    client_code(car)
    print("")
    print("Launched with the Plane.")
    client_code(Plane)
    Plane.refueling(100)
    Plane.updateweather("Хорошая")
    client_code(Plane)
```

proxy.py

```
from abc import ABC, abstractmethod
from contextlib import contextmanager
from datetime import datetime
from time import sleep

allowusers = ['maxzbox', "andrey-kireev", "valerdon"]


@contextmanager
def workwithfile():
    a = open("../log", 'a')
    yield a
```

```
    a.close()


class Subject(ABC):
    @abstractmethod
    def request(self, login) -> None:
        pass


class RealSubject(Subject):
    def request(self, login, request):
        print("RealSubject: Handling request.")
        return ("some data")


class Proxy(Subject):

    def __init__(self, real_subject: RealSubject) -> None:
        self._real_subject = real_subject

    def request(self, login, request) -> None:

        if self.check_access(login):
            self.log_access(f"User {login} Logging successfully request: {request}")
            return(self._real_subject.request(login, request))
        else:
            self.log_access(f"No rights to enter, attempted by {login} with request:
{request}")
            print("no permission")
            return None

    def check_access(self, login) -> bool:
        print("Proxy: Checking access prior to firing a real request.")
        if (login in allowusers):
            return True
        else:
            return False

    @staticmethod
    def log_access(massage) -> None:
        with workwithfile() as f:
            f.write(f"{datetime.isoformat(datetime.now())}- {massage}\n")


def client_code(subject: Subject) -> None:
    subject.request('user', "select * from passwd")
    sleep(0.2)
    subject.request('maxzbox', “select * from todo”)


if __name__ == “__main__”:
```

```
    print("Client: Executing the client code with a real subject:")
    real_subject = RealSubject()
    client_code(real_subject)

    print("")

    print("Client: Executing the same client code with a proxy:")
    proxy = Proxy(real_subject)
    client_code(proxy)
```

strategy.py

```
from __future__ import annotations
from abc import ABC, abstractmethod

trace1 = ['A', 'B', 'C', 'D', 'E', 'F']
trace2 = ['L', 'E12', 'D16', 'C', 'D', 'E', 'F']
placebyindex = {'A': 0, 'B': 1, 'C' :2, 'D': 3, 'E': 4, 'F': 5}
Busess = {"busA": trace1, "busB": trace2}
class Navigator():
    def __init__(self, strategy: Strategy) -> None:
        """
        Обычно Контекст принимает стратегию через конструктор, а также
        предоставляет сеттер для её изменения во время выполнения.
        """

        self._strategy = strategy

    @property
    def strategy(self) -> Strategy:
        return self._strategy

    @strategy.setter
    def strategy(self, strategy: Strategy) -> None:
        self._strategy = strategy

    def do_some_business_logic(self, start, end):
        result = self._strategy.do_algorithm(start, end)
        return result


class Strategy(ABC):
    @abstractmethod
    def do_algorithm(self, startpoint, pointend):
        pass

class Strategypublictransport(Strategy):
    def do_algorithm(self, pointstart, pointend):
        a = [I for I in Busess if pointstart in Busess[i] and pointend in Busess[i]]
        if len(a) > 0:
            print(f"trace from {pointstart} to {pointend} could be completed by publi
c transport with such buses:{a}")
```

```
            return a
        else:
            print("no trace by public transport")
            return None


class StrategybyFoot(Strategy):
    def __init__(self):
        self._matrix = [0] * 6
        self._matrix[0] = 1
        self._matrix[1] = 2
        self._matrix[2] = 3

    def do_algorithm(self, startpoint, endpoint):
        if startpoint in placebyindex.keys() and endpoint in placebyindex.keys():
            index1 = placebyindex[startpoint]
            index2 = placebyindex[endpoint]
            if self._matrix[index1] == index2:
                print(f"you can go by foot from place {startpoint } to {endpoint} by
foot")
                return f"you can go by foot from place {startpoint } to {endpoint} by
 foot"
            else:
                print("no way by Foot")
                return "no way by Foot"

        else:
            print("Sorry no place in map")
            return None


if __name__ == "__main__":
    # Клиентский код выбирает конкретную стратегию и передаёт её в контекст.
    # Клиент должен знать о различиях между стратегиями, чтобы сделать
    # правильный выбор.

    Context = Navigator(Strategypublictransport())
    print("Client: Strategy is set to public transport.")
    context.do_some_business_logic('A', 'B')
    print()

    print("Client: Strategy is set to reverse sorting.")
    context.strategy = StrategybyFoot()
    context.do_some_business_logic('A', 'R')
```

4. Результат выполнения

```
PS D:\Документы\РИП> & C:/Users/User/AppData/Local/Programs/Python/Python38-32/python.exe d:/Документы/РИП/Lab4/patterns/factorymethod.py
Launched with the car.
Client: I'm don't now who is deliver exactly.
DELIVER: delivering can't be done no fuel.

Launched with the Plane.
Client: I'm don't now who is deliver exactly.
DELIVER: delivering can't be done no fuel.
Client: I'm don't now who is deliver exactly.
DELIVER: The product Coal are delivered
PS D:\Документы\РИП> & C:/Users/User/AppData/Local/Programs/Python/Python38-32/python.exe d:/Документы/РИП/Lab4/patterns/proxy.py
Client: Executing the client code with a real subject:
RealSubject: Handling request.
RealSubject: Handling request.

Client: Executing the same client code with a proxy:
Proxy: Checking access prior to firing a real request.
no permission
Proxy: Checking access prior to firing a real request.
RealSubject: Handling request.
PS D:\Документы\РИП> & C:/Users/User/AppData/Local/Programs/Python/Python38-32/python.exe d:/Документы/РИП/Lab4/patterns/strategy.py
Client: Strategy is set to public transport.
trace from A to B could be completed by public transport with such buses:['busA']

Client: Strategy is set to reverse sorting.
Sorry no place in map
```