Московский государственный технический университет имени Н. Э. Баумана

Факультет «Информатика и системы управления»

Кафедра ИУ5

Отчёт по
лабораторной работе № 4
«Разработка интернет приложений»

Подготовил:

Кан Андрей Дмитриевич

Группа ИУ5-54Б

Подпись_____

Дата_____

Москва
2020г.

1.) 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.

2.) Текст программы

Тесты:

```
def test_charging():
    iphone = Iphone()
    lightningwire = LightningWire()
    usbwire = UsbWire()
    adapterusb = AdapterUsb(usbwire)

    assert iphone.charge(lightningwire)
    assert not iphone.charge(usbwire)
    assert iphone.charge(adapterusb)

def get_cheese_list():
    cheese_list = [Cheese(), Cheese(), Cheese()]
    return cheese_list


def test_fabric(monkeypatch):
    cheesefabric = CheeseFabric()
    monkeypatch.setattr(cheesefabric, "deliver", get_cheese_list)
    assert len(cheesefabric.deliver()) == 3
    sourcreamefabric = SourCreameFabric()
    assert type(sourcreamefabric.deliver(2)) == list
    assert len(sourcreamefabric.deliver(2)) == 2
    assert type(sourcreamefabric.deliver(2)[0]) == SourCreame

def test_bases():
    elf_base = ElfBaseAI(2000)
    assert elf_base.gather_army() == "5 elves were recruited"
    assert elf_base.build_structures() == "2 structures were built"
    orc_base = OrcBaseAI(3000)
    assert orc_base.gather_army() == "20 orcs were recruited"
    assert orc_base.build_structures() == "3 structures were built"
```

Классы:

```
class AdapterUsb(LightningWire):
    def __init__(self, usbwire: UsbWire):
        self.usbwire = usbwire

    def get_port(self) -> str:
```

```
        if self.usbwire.get_port() == "usb": # если разъемы переходника и кабеля совпадают, то мы соединяем переходник и получаем другой разъем на выходе
            return "lightning"
        else:
            return "incompatible ports"

class LightningWire:

    def __init__(self):
        self.__port = "lightning"

    def get_port(self) -> str:
        return self.__port

import time


class Iphone:
    def __init__(self):
        self.__port = "lightning"

    def charge(self, wire: LightningWire):
        if self.__port == wire.get_port():
            print("Charging...")
            time.sleep(1)
            print("Your iphone is fully charged")
            return True
        else:
            print("Incompatible ports")
            return False

class UsbWire:

    def __init__(self):
        self.__port = "usb"

    def get_port(self):
        return self.__port

class MilkFabric(ABC):

    @abstractmethod
    def create_milk_product(self) -> MilkProduct:
        pass

    def deliver(self, amount: int) -> list[MilkProduct]:
        products = []
        for i in range(amount):
            products.append(self.create_milk_product())
        print("Products with code name {} were successfully delivered".format(products[0]))
        return products


class CheeseFabric(MilkFabric):

    def create_milk_product(self) -> MilkProduct:
        return Cheese()
```

```
class SourCreameFabric(MilkFabric):

    def create_milk_product(self) -> MilkProduct:
        return SourCreame()

class MilkProduct(ABC):

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


class Cheese(MilkProduct):

    def __repr__(self) -> str:
        return "Cheese"


class SourCreame(MilkProduct):

    def __repr__(self) -> str:
        return "SourCreame"

class BaseAI(ABC):
    """Base class"""

    @abstractmethod
    def build_structures(self):
        pass

    @abstractmethod
    def gather_army(self):
        pass

    def attack(self, target: BaseAI):
        """default method"""
        return "Attacking {}".format(target)

    def turn(self, target: BaseAI):
        print(self.build_structures())
        print(self.gather_army())
        print(self.attack(target))


class ElfBaseAI(BaseAI):
    """Elves are more advanced and prefer to spend more money on buildings
    but their army and buildings cost more"""

    def __init__(self, money):
        self.__money = money
        self.__unit = Elf()
        self.__building_cost = 500
        self.built_structures = 0
        self.army = []
        self.__unit_cost = 200

    def build_structures(self):
        amount = int((self.__money/2) / self.__building_cost)
        self.built_structures = amount
```

```
        return "{} structures were built".format(self.built_structures)

    def gather_army(self):
        amount = int((self.__money/2)/self.__unit_cost)
        for i in range(amount):
            self.army.append(Elf())
        return "{} elves were recruited".format(len(self.army))

    def __repr__(self):
        return "ElfBase"


class OrcBaseAI(BaseAI):
    """Orcs prefer unreasonable fights to tactic moves that's why they
    like to spend money on army"""

    def __init__(self, money):
        self.__money = money
        self.__unit = Orc()
        self.__building_cost = 300
        self.built_structures = 0
        self.army = []
        self.__unit_cost = 100

    def build_structures(self):
        amount = int((self.__money / 3) / self.__building_cost)
        self.built_structures = amount
        return "{} structures were built".format(self.built_structures)

    def gather_army(self):
        amount = int((self.__money * 2 / 3) / self.__unit_cost)
        for i in range(amount):
            self.army.append(Elf())
        return "{} orcs were recruited".format(len(self.army))

    def __repr__(self):
        return "OrcBase"


class Unit(ABC):

    @abstractmethod
    def __repr__(self):
        pass


class Elf(Unit):
    def __init__(self):
        self.__unit = "elf"

    def __repr__(self):
        return self.__unit


class Orc(Unit):
    def __init__(self):
        self.__unit = "orc"

    def __repr__(self):
```

```
        return self.__unit
```

Main:

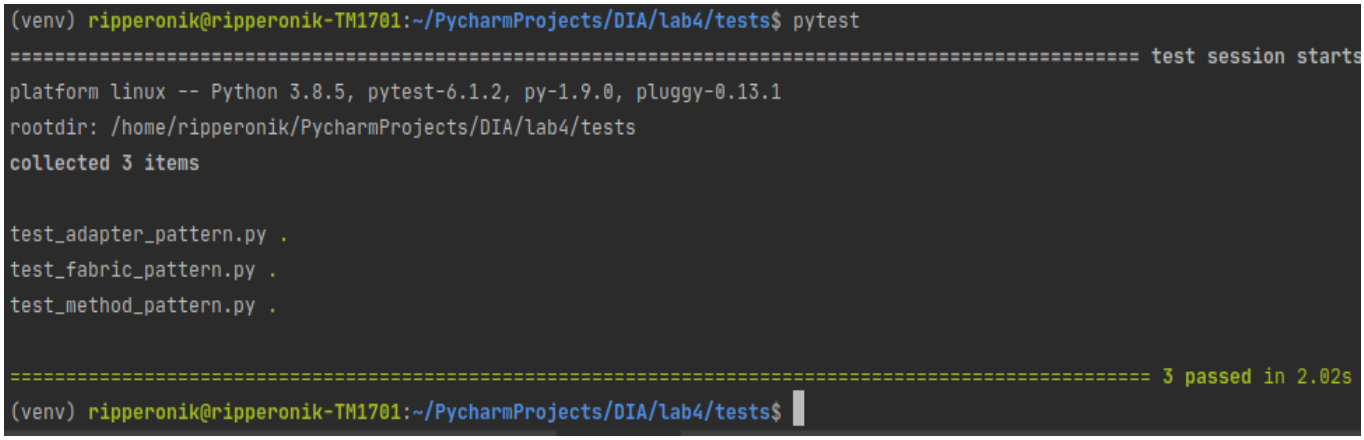
```
from patterns.fabric_pattern.MilkFabric import CheeseFabric, SourCreameFabric
from patterns.adapter_pattern.Smartphone import Iphone
from patterns.adapter_pattern.LightningWire import LightningWire
from patterns.adapter_pattern.AdapterUsb import AdapterUsb
from patterns.adapter_pattern.UsbWire import UsbWire
from patterns.method_pattern.GameAI import ElfBaseAI, OrcBaseAI

if __name__ == '__main__':
    cheeseFabric = CheeseFabric()
    print(cheeseFabric.deliver(2))
    sourcreameFabric = SourCreameFabric()
    print(sourcreameFabric.deliver(3))

    iphone = Iphone()
    lightningwire = LightningWire()
    usbwire = UsbWire()
    adapterusb = AdapterUsb(usbwire)
    print(iphone.charge(lightningwire))
    print(iphone.charge(adapterusb))
    print(iphone.charge(usbwire))

    elfbase = ElfBaseAI(2000)
    orcbase = OrcBaseAI(2000)
    elfbase.turn(orcbase)
    orcbase.turn(elfbase)
```

3.) Экранные формы с результатами выполнения задания



Результаты работы main:
Products with code name Cheese were successfully delivered
[Cheese, Cheese]
Products with code name SourCreame were successfully delivered
[SourCreame, SourCreame, SourCreame]

Charging...
Your iphone is fully charged
True
Charging...
Your iphone is fully charged
True
Incompatible ports
False
2 structures were built
5 elves were recruited
Attacking OrcBase
2 structures were built
13 orcs were recruited
Attacking ElfBase