МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н.Э.Баумана

Отчет по лабораторной работе №4

по курсу «Разработка интернет-приложений»

Шаблоны проектирования и модульное тестирование в Python

Подготовил

Ионов С.А.

ИУ5-52Б

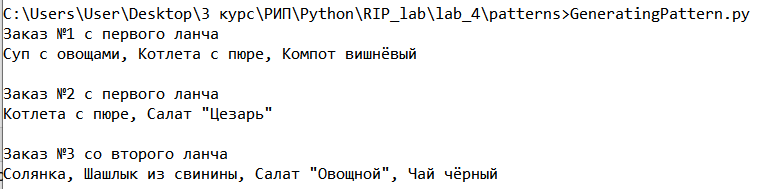
1. Описание задания
2. Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать [следующий каталог.](https://refactoring.guru/ru/design-patterns/catalog)
3. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк.
   * BDD - фреймворк.
   * Создание Mock-объектов.
4. Текст программы
5. **Порождающий шаблон проектирования (GeneratingPattern.py)**

В качестве порождающего шаблона используется паттерн строитель, который позволяет создавать сложные объекты пошагово, вызывая только те объекты, которые нужны. В данной задаче приведен пример применения этого паттерна к предметной области «Заказ еды в кафе». Строительство заключается в построении заказа из отдельных блюд, принадлежащих конкретному типу ланча (конкретному строителю).

Текст программы:

**from** abc **import** ABC, abstractmethod, abstractproperty  
  
**class** Business\_lunch(ABC):  
 *"""  
 Интерфейс Строителя объявляет создающие методы для различных частей объектов  
 Продуктов.  
 """* @abstractproperty  
 **def** lunch(self):  
 *"""Продуктом является ланч."""* **pass** @abstractmethod  
 **def** add\_first\_dish(self):  
 **pass** @abstractmethod  
 **def** add\_second\_dish(self):  
 **pass** @abstractmethod  
 **def** add\_salad(self):  
 **pass** @abstractmethod  
 **def** add\_drink(self):  
 **pass  
  
class** Business\_lunch\_1(Business\_lunch):  
 *"""Конкретный строитель, строящий ланч первого типа."""* **def** \_\_init\_\_(self):  
 self.reset()  
  
 **def** reset(self):  
 self.\_lunch = Lunch()  
  
 @property  
 **def** lunch(self):  
 lunch = self.\_lunch  
 **return** lunch  
  
 **def** add\_first\_dish(self):  
 self.\_lunch.add(**"Суп с овощами"**, 100)  
  
 **def** add\_second\_dish(self):  
 self.\_lunch.add(**"Котлета с пюре"**, 150)  
  
 **def** add\_salad(self):  
 self.\_lunch.add(**"Салат \"Цезарь\""**, 125)  
  
 **def** add\_drink(self):  
 self.\_lunch.add(**"Компот вишнёвый"**, 90)  
  
 **def** add\_all(self):  
 self.add\_first\_dish()  
 self.add\_second\_dish()  
 self.add\_salad()  
 self.add\_drink()  
  
  
**class** Business\_lunch\_2(Business\_lunch):  
 *"""Конкретный строитель, строящий ланч второго типа."""* **def** \_\_init\_\_(self):  
 self.reset()  
  
 **def** reset(self):  
 self.\_lunch = Lunch()  
  
 @property  
 **def** lunch(self):  
 lunch = self.\_lunch  
 **return** lunch  
  
 **def** add\_first\_dish(self):  
 self.\_lunch.add(**"Солянка"**, 130)  
  
 **def** add\_second\_dish(self):  
 self.\_lunch.add(**"Шашлык из свинины"**, 225)  
  
 **def** add\_salad(self):  
 self.\_lunch.add(**"Салат \"Овощной\""**, 100)  
  
 **def** add\_drink(self):  
 self.\_lunch.add(**"Чай чёрный"**, 60)  
  
 **def** add\_all(self):  
 self.add\_first\_dish()  
 self.add\_second\_dish()  
 self.add\_salad()  
 self.add\_drink()  
  
  
**class** Lunch():  
 **def** \_\_init\_\_(self):  
 self.lunch = []  
 self.sum = 0  
  
 **def** add(self, dish, price):  
 self.lunch.append(dish)  
 self.sum += price  
  
 **def** list\_lunch(self):  
 **return f"{', '**.join(self.lunch)**}"  
  
 def** get\_sum(self):  
 **return** self.sum  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 print(**'Заказ №1 с первого ланча'**)  
 order = Business\_lunch\_1()  
 order.add\_first\_dish()  
 order.add\_second\_dish()  
 order.add\_drink()  
 print(order.lunch.list\_lunch())  
  
 print(**'\nЗаказ №2 с первого ланча'**)  
 order.reset()  
 order.add\_second\_dish()  
 order.add\_salad()  
 print(order.lunch.list\_lunch())  
  
 print(**'\nЗаказ №3 со второго ланча'**)  
 order = Business\_lunch\_2()  
 order.add\_all()  
 print(order.lunch.list\_lunch())

Экранные формы:



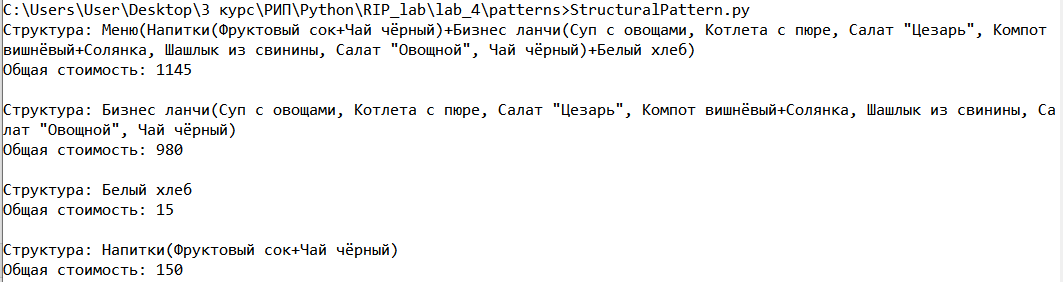
1. **Структурный шаблон проектирования (StructuralPattern.py)**

В качестве структурного шаблона используется паттерн компоновщик, который позволяет сгруппировать множество объектов в древовидную структуру, а затем работать с ней как с единичным объектом. В данной задаче приведен пример применения этого паттерна к предметной области «Меню в кафе». Меню состоит из компонентов, которые в свою очередь тоже могут состоять из более простейших объектов. В конечном счете, мы приходим к элементарных объектам, которые являются листьями дерева. Так мы получаем древовидную структуру. Компоновщик позволяет рассматривать объекты и элементарные объекты через единый интерфейсом. В данной задаче нам необходимо узнать стоимость определенных компонентов меню. В этом случае элементарный объект просто возвращает свою цену, а объект суммирует цены всех входящих в него объектов.

Текст программы:

**from** abc **import** ABC, abstractmethod  
**from** patterns.GeneratingPattern **import** Business\_lunch\_1, Business\_lunch\_2  
  
  
**class** Component(ABC):  
 *"""  
 Базовый класс Компонент объявляет общие операции как для простых, так и для  
 сложных объектов структуры.  
 """* @property  
 **def** parent(self):  
 **return** self.\_parent  
  
 @parent.setter  
 **def** parent(self, parent):  
 self.\_parent = parent  
  
 **def** add(self, component):  
 **pass  
  
 def** remove(self, component):  
 **pass  
  
 def** is\_composite(self):  
 **return False** @abstractmethod  
 **def** operation(self):  
 **pass** @abstractmethod  
 **def** get\_price(self):  
 **pass  
  
  
class** Leaf(Component):  
 *"""Конечный объект, не имеющий вложенных."""* **def** \_\_init\_\_(self, value, price):  
 self.\_value = value  
 self.\_price = price  
  
 **def** operation(self):  
 **return** self.\_value  
  
 **def** get\_price(self):  
 **return** self.\_price  
  
  
**class** Composite(Component):  
 *"""Объект, имеющий вложенные объекты."""* **def** \_\_init\_\_(self, name):  
 self.\_children = []  
 self.\_name = name  
  
 **def** add(self, component):  
 self.\_children.append(component)  
 component.parent = self  
  
 **def** remove(self, component):  
 self.\_children.remove(component)  
 component.parent = **None  
  
 def** is\_composite(self):  
 **return True  
  
 def** operation(self):  
 results = []  
 **for** child **in** self.\_children:  
 results.append(child.operation())  
 **return** self.\_name+**f"({'+'**.join(results)**})"  
  
 def** get\_price(self):  
 count = 0  
 **for** child **in** self.\_children:  
 count += child.get\_price()  
 **return** count  
  
  
**def** client\_code(component):  
 print(**f"Структура: {**component.operation()**}"**)  
 print(**f'Общая стоимость: {**component.get\_price()**}'**, end=**'\n\n'**)  
  
  
**if** \_\_name\_\_==**'\_\_main\_\_'**:  
 menu = Composite(**'Меню'**)  
 drinks = Composite(**'Напитки'**)  
 drinks.add(Leaf(**'Фруктовый сок'**, 100))  
 drinks.add(Leaf(**'Чай чёрный'**, 50))  
 business\_lunches = Composite(**'Бизнес ланчи'**)  
 lunch1 = Business\_lunch\_1()  
 lunch1.add\_all()  
 lunch2 = Business\_lunch\_2()  
 lunch2.add\_all()  
 business\_lunches.add(Leaf(lunch1.lunch.list\_lunch(), lunch1.lunch.get\_sum()))  
 business\_lunches.add(Leaf(lunch2.lunch.list\_lunch(), lunch2.lunch.get\_sum()))  
 white\_bread = Leaf(**'Белый хлеб'**, 15)  
  
 menu.add(drinks)  
 menu.add(business\_lunches)  
 menu.add(white\_bread)  
  
 client\_code(menu)  
 client\_code(business\_lunches)  
 client\_code(white\_bread)  
 client\_code(drinks)

Экранные формы:



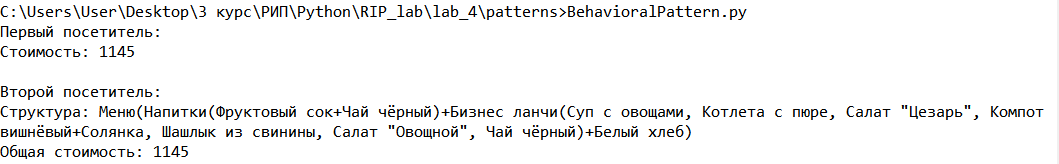
1. **Поведенческий шаблон проектирования (BehavioralPattern.py)**

В качестве поведенческого шаблона используется паттерн посетитель, который позволяет добавлять в программу новые операции, не изменяя классы объектов, над которыми эти операции могут выполняться. В данной задаче приведен пример применения этого паттерна к различному выводу информации о меню. Первый посетитель выводит только общую стоимость меню, а второй – стоимость и структуру меню.

Текст программы:

**from** patterns.StructuralPattern **import** Composite, Component, Leaf, client\_code  
**from** patterns.GeneratingPattern **import** Business\_lunch\_1, Business\_lunch\_2  
**from** abc **import** ABC, abstractmethod  
  
  
**class** ComponentNew(Component):  
 *"""  
 Интерфейс Компонента объявляет метод accept, который в качестве аргумента  
 может получать любой объект, реализующий интерфейс посетителя.  
 """* @abstractmethod  
 **def** accept(self, visitor):  
 **pass  
  
class** CompositeNew(Composite, ComponentNew):  
 **def** accept(self, visitor):  
 visitor.visit\_component(self)  
  
**class** Visitor(ABC):  
 @abstractmethod  
 **def** visit\_component(self, element):  
 **pass  
  
class** Visitor1(Visitor):  
 **def** visit\_component(self, element):  
 print(**'Стоимость: {}'**.format(element.get\_price()))  
  
**class** Visitor2(Visitor):  
 **def** visit\_component(self, element):  
 client\_code(element)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 menu = CompositeNew(**'Меню'**)  
 drinks = Composite(**'Напитки'**)  
 drinks.add(Leaf(**'Фруктовый сок'**, 100))  
 drinks.add(Leaf(**'Чай чёрный'**, 50))  
 business\_lunches = Composite(**'Бизнес ланчи'**)  
 lunch1 = Business\_lunch\_1()  
 lunch1.add\_all()  
 lunch2 = Business\_lunch\_2()  
 lunch2.add\_all()  
 business\_lunches.add(Leaf(lunch1.lunch.list\_lunch(), lunch1.lunch.get\_sum()))  
 business\_lunches.add(Leaf(lunch2.lunch.list\_lunch(), lunch2.lunch.get\_sum()))  
 white\_bread = Leaf(**'Белый хлеб'**, 15)  
  
 menu.add(drinks)  
 menu.add(business\_lunches)  
 menu.add(white\_bread)  
  
 visitor1 = Visitor1()  
 visitor2 = Visitor2()  
 print(**"Первый посетитель:"**)  
 menu.accept(visitor1)  
 print(**"\nВторой посетитель:"**)  
 menu.accept(visitor2)

Экранные формы:



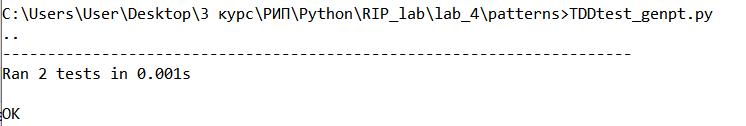
1. **TDD-фреймворк (TDDtest\_genpt.py)**

В качестве TDD-фреймворка используется модуль unittest, который используется для тестирования подсчета суммы бизнес-ланчей в модуле, реализующем порождающий шаблон проектирования. В данном случае методы рассчитывают правильно.

Текст программы:

**from** patterns.GeneratingPattern **import** Business\_lunch\_1, Business\_lunch\_2  
**import** unittest  
  
  
**class** sumTest(unittest.TestCase):  
 **def** test\_sum\_Blunch1(self):  
 order = Business\_lunch\_1()  
 order.add\_all()  
 self.assertEqual(order.lunch.get\_sum(), 465, **"Should be 465"**)  
  
 **def** test\_sum\_Blunch2(self):  
 order = Business\_lunch\_2()  
 order.add\_all()  
 self.assertEqual(order.lunch.get\_sum(), 515, **"Should be 515"**)  
  
**if** \_\_name\_\_==**"\_\_main\_\_"**:  
 unittest.main()

Экранные формы:



1. **BDD-фреймворк (strpt.feature, BDDtest\_strpt.py)**

Текст программы:

**Feature**: My first feature file using radish  
 In order to test my awesome software  
 I need an awesome BDD tool like radish  
 to test my software.  
  
 **Scenario**: Test my function get\_price()  
 **Given** I have the component drinks = Composite('Напитки') drinks.add(Leaf('Фруктовый сок', 100)) drinks.add(Leaf('Чай чёрный', 50))  
 **When** I get price from them  
 **Then** I expect the result to be 150

**from** radish **import** given, when, then  
  
  
@given(**"I have the component {component1: g}"**)  
**def** have\_component(step, component1):  
 step.context.component1 = component1  
  
@when(**"I get price from them"**)  
**def** get\_price\_component(step):  
 step.context.result = step.context.component1.get\_price()  
  
@then(**"I expect the result to be {result: g}"**)  
**def** expect\_result(step, result):  
 **assert** step.context.result == result

1. **TDD-фреймворк (TDDtest\_bhv.py)**

Текст программы:

**from** patterns.BehavioralPattern **import** \*  
**import** unittest  
  
  
**class** sumTest(unittest.TestCase):  
 **def** test\_sum\_menu(self):  
 menu = CompositeNew(**'Меню'**)  
 drinks = Composite(**'Напитки'**)  
 drinks.add(Leaf(**'Фруктовый сок'**, 100))  
 drinks.add(Leaf(**'Чай чёрный'**, 50))  
 business\_lunches = Composite(**'Бизнес ланчи'**)  
 lunch1 = Business\_lunch\_1()  
 lunch1.add\_all()  
 lunch2 = Business\_lunch\_2()  
 lunch2.add\_all()  
 business\_lunches.add(Leaf(lunch1.lunch.list\_lunch(), lunch1.lunch.get\_sum()))  
 business\_lunches.add(Leaf(lunch2.lunch.list\_lunch(), lunch2.lunch.get\_sum()))  
 white\_bread = Leaf(**'Белый хлеб'**, 15)  
  
 menu.add(drinks)  
 menu.add(business\_lunches)  
 menu.add(white\_bread)  
  
 visitor1 = Visitor1()  
 self.assertEqual(menu.accept(visitor1), **'visitor\_for\_composite\_new'**, **"Should be 'visitor\_for\_composite\_new'"**)  
  
  
**if** \_\_name\_\_==**"\_\_main\_\_"**:  
 unittest.main()

Экранные формы:

