**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

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

**(МГТУ им. Н.Э. Баумана)**

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **«Информатики и систем управления»** |
| **КАФЕДРА** | Системы обработки информации и управления |

Дисциплина «Разработка Интернет-Приложений»

**ЛАБОРАТОРНАЯ РАБОТА №4**

Шаблоны проектирования и модульное тестирование в Python

|  |  |
| --- | --- |
| Студент | Сахарова Е. К. ИУ5-52Б |
| Преподаватель | Гапанюк Ю. Е. |

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

**Задание:**

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

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

**Код программы:**

Предметная область для выполнения задания – Торт.

builder.py

**from** abc **import** ABCMeta, abstractstaticmethod  
  
  
**class** ICakeBuilder(metaclass=ABCMeta):  
 *"""Интерфейс строителя"""* @abstractstaticmethod  
 **def** set\_foundation(self, value):  
 *"""Задает основу для торта"""* @abstractstaticmethod  
 **def** set\_shape\_type(self, value):  
 *"""Задает форму для торта"""* @abstractstaticmethod  
 **def** set\_filling(self, value):  
 *"""Задает тип начинки"""* @abstractstaticmethod  
 **def** set\_number\_tiers(self, value):  
 *"""Задает количество ярусов торта"""* @abstractstaticmethod  
 **def** get\_result(self):  
 *"""Возвращает торт"""***class** CakeBuilder(ICakeBuilder):  
 *"""Конкретный строитель"""* **def** \_\_init\_\_(self):  
 self.cake = Cake()  
  
 **def** set\_foundation(self, value):  
 self.cake.foundation = value  
 **return** self  
  
 **def** set\_shape\_type(self, value):  
 self.cake.shape\_type = value  
 **return** self  
  
 **def** set\_filling(self, value):  
 self.cake.filling = value  
 **return** self  
  
 **def** set\_number\_tiers(self, value):  
 self.cake.tiers = value  
 **return** self  
  
 **def** get\_result(self):  
 **return** self.cake  
  
  
**class** Cake:  
 *"""Торт"""* **def** \_\_init\_\_(self, foundation=**'бисквитный'**, shape\_type=**'круга'**, filling=**'шоколадная'**, tiers=1):  
 *# бисквитный, песочный, слоеный, шоколадный* self.foundation = foundation  
 *# круг, овал, квадрат, сердце, ромб, звезда* self.shape\_type = shape\_type  
 *# шоколадная, клубничная, вишневая, йогуртовая, ореховая* self.filling = filling  
 self.tiers = tiers  
  
 **def** \_\_str\_\_(self):  
 **return 'Это {0} торт в форме {1} с {2} начинкой и {3} ярусами.'**.format(  
 self.foundation, self.shape\_type, self.filling, self.tiers  
 )  
  
  
**class** YogurtDirector:  
 @staticmethod  
 **def** construct():  
 **return** CakeBuilder()\  
 .set\_foundation(**'бисквитный'**)\  
 .set\_shape\_type(**'овала'**)\  
 .set\_filling(**'йогуртовой'**)\  
 .set\_number\_tiers(3)\  
 .get\_result()  
  
  
**class** StrawberryDirector:  
 @staticmethod  
 **def** construct():  
 **return** CakeBuilder()\  
 .set\_foundation(**'бисквитный'**)\  
 .set\_shape\_type(**'сердца'**)\  
 .set\_filling(**'клубничной'**)\  
 .set\_number\_tiers(1)\  
 .get\_result()  
  
  
**class** ChocolateDirector:  
 @staticmethod  
 **def** construct():  
 **return** CakeBuilder()\  
 .set\_foundation(**'шоколадный'**)\  
 .set\_shape\_type(**'квадрата'**)\  
 .set\_filling(**'ореховой'**)\  
 .set\_number\_tiers(1)\  
 .get\_result()  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 YOGURT\_CAKE = YogurtDirector.construct()  
 print(YOGURT\_CAKE)  
 STRAWBERRY\_CAKE = StrawberryDirector.construct()  
 print(STRAWBERRY\_CAKE)  
 CHOCOLATE\_CAKE = ChocolateDirector.construct()  
 print(CHOCOLATE\_CAKE)

test-tdd.py

**import** unittest  
**import** sys  
**import** os  
*# from github.lab4.builder import \****from** builder **import** \*  
  
  
sys.path.append(os.getcwd())  
  
  
**class** TestFoundation(unittest.TestCase):  
 **def** test\_yogurt\_cake\_returns\_string(self):  
 self.assertEqual(**'{}'**.format(YogurtDirector.construct()),  
 **'Это бисквитный торт в форме овала с йогуртовой начинкой и 3 ярусами.'**)  
  
 **def** test\_strawberry\_cake\_returns\_string(self):  
 self.assertEqual(**'{}'**.format(StrawberryDirector.construct()),  
 **'Это бисквитный торт в форме сердца с клубничной начинкой и 1 ярусами.'**)  
  
 **def** test\_chocolate\_cake\_returns\_string(self):  
 self.assertEqual(**'{}'**.format(ChocolateDirector.construct()),  
 **'Это шоколадный торт в форме квадрата с ореховой начинкой и 1 ярусами.'**)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 unittest.main()

bridge.py

**from** abc **import** ABCMeta, abstractmethod  
  
  
**class** Decoration(metaclass=ABCMeta):  
 *"""Декорирование"""* @abstractmethod  
 **def** decorating\_strawberry\_cake(self, items):  
 *"""Клубничный торт"""* @abstractmethod  
 **def** decorating\_chocolate\_cake(self, items):  
 *"""Шоколадный торт"""***class** Candle(Decoration):  
 *"""Декорирование свечами"""* **def** decorating\_strawberry\_cake(self, items):  
 **return f"Клубничный торт с {**items**} свечами."  
  
 def** decorating\_chocolate\_cake(self, items):  
 **return f"Шоколадный торт с {**items**} свечами."  
  
  
class** Flower(Decoration):  
 *"""Декорирование цветами"""* **def** decorating\_strawberry\_cake(self, items):  
 **return f"Клубничный торт с {**items**} цветами."  
  
 def** decorating\_chocolate\_cake(self, items):  
 **return f"Шоколадный торт с {**items**} цветами."  
  
  
class** Cake(metaclass=ABCMeta):  
 *"""Торт"""* @abstractmethod  
 **def** \_\_init\_\_(self, decoration):  
 self.decoration = decoration  
  
 @abstractmethod  
 **def** display\_description(self):  
 *"""Выводит информацию"""* @abstractmethod  
 **def** add\_objects(self):  
 *"""Добавляет объект декора"""***class** Strawberry(Cake):  
 **def** \_\_init\_\_(self, decoration, objects=0):  
 super().\_\_init\_\_(decoration)  
 self.objects = objects  
  
 **def** display\_description(self):  
 **return** self.decoration.decorating\_strawberry\_cake(self.objects)  
  
 **def** add\_objects(self, new\_objects):  
 self.objects += new\_objects  
  
  
**class** Chocolate(Cake):  
 **def** \_\_init\_\_(self, decoration, objects=0):  
 super().\_\_init\_\_(decoration)  
 self.objects = objects  
  
 **def** display\_description(self):  
 **return** self.decoration.decorating\_chocolate\_cake(self.objects)  
  
 **def** add\_objects(self, new\_objects):  
 self.objects += new\_objects  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 candle = Candle()  
 flower = Flower()  
  
 chocolate1 = Chocolate(candle, 20)  
 chocolate2 = Chocolate(flower, 2)  
 strawberry1 = Strawberry(candle, 4)  
 strawberry2 = Strawberry(flower, 50)  
  
 print(strawberry1.display\_description())  
 strawberry1.add\_objects(1)  
 print(strawberry1.display\_description())  
  
 print(chocolate1.display\_description())  
 print(chocolate2.display\_description())  
 print(strawberry2.display\_description())

cake.feature

Feature: Test Bridge Design Pattern Functionality  
 Scenario: Strawberry Cake with 8 Candles  
 Given Bridge app is run  
 When I input "4" candles to put on the cake  
 Then I get result "Клубничный торт с 4 свечами."

steps.py

**from** behave **import** given, when, then  
**from** bridge **import** \*  
  
  
@given(**u'Bridge app is run'**)  
**def** step\_impl(context):  
 print(**u'STEP: Given Bridge app is run'**)  
 **pass**@when(**u'I input "{inp}" candles to put on the cake'**)  
**def** step\_impl(context, inp):  
 print(**u'STEP: When I input "{}" to put on the cake'**.format(inp))  
 context.result = **'{}'**.format(Strawberry(Candle(), inp).display\_description())  
  
  
@then(**u'I get result "{out}"'**)  
**def** step\_impl(context, out):  
 print(**u'STEP: Then I get result "{}"'**.format(out))  
 **assert** context.result == out, **'Expected {}, got {}'**.format(out, context.result)

chain\_of\_responsibility.py

**from** abc **import** ABCMeta, abstractstaticmethod  
  
  
**class** ICake(metaclass=ABCMeta):  
 *"""Интерфейс обработки запросов"""* @abstractstaticmethod  
 **def** set\_successor(successor):  
 *"""Установка следующего обработчика в цепочке"""* @abstractstaticmethod  
 **def** handle(amount):  
 *"""Обработка события"""***class** BigCake(ICake):  
 *"""Конкретный торт  
 Выдает большой торт весом 5 кг, если применимо,  
 в противном случае продолжает """* **def** \_\_init\_\_(self):  
 self.\_successor = **None  
  
 def** set\_successor(self, successor):  
 *"""Установка преемника"""* self.\_successor = successor  
  
 **def** handle(self, amount):  
 *"""Обработка торта"""* **if** amount >= 5:  
 num = amount // 5  
 remainder = amount % 5  
 print(**f'Больших тортов весом 5 кг -- {**num**}'**)  
 **if** remainder != 0:  
 self.\_successor.handle(remainder)  
 **else**:  
 self.\_successor.handle(amount)  
  
  
**class** MediumCake(ICake):  
 *"""Конкретный торт  
 Выдает большой торт весом 3 кг, если применимо,  
 в противном случае продолжает """* **def** \_\_init\_\_(self):  
 self.\_successor = **None  
  
 def** set\_successor(self, successor):  
 *"""Установка преемника"""* self.\_successor = successor  
  
 **def** handle(self, amount):  
 *"""Обработка торта"""* **if** amount >= 3:  
 num = amount // 3  
 remainder = amount % 3  
 print(**f'Средних тортов весом 3 кг -- {**num**}'**)  
 **if** remainder != 0:  
 self.\_successor.handle(remainder)  
 **else**:  
 self.\_successor.handle(amount)  
  
  
**class** SmallCake(ICake):  
 *"""Конкретный торт  
 Выдает большой торт весом 1 кг, если применимо,  
 в противном случае продолжает """* **def** \_\_init\_\_(self):  
 self.\_successor = **None  
  
 def** set\_successor(self, successor):  
 *"""Установка преемника"""* self.\_successor = successor  
  
 **def** handle(self, amount):  
 *"""Обработка торта"""* **if** amount >= 1:  
 num = amount // 1  
 remainder = amount % 1  
 print(**f'Маленьких тортов весом 1 кг -- {**num**}'**)  
 **if** remainder != 0:  
 self.\_successor.handle(remainder)  
 **else**:  
 self.\_successor.handle(amount)  
  
  
**class** CakeChain:  
 *"""Связь тортов"""* **def** \_\_init\_\_(self):  
 self.chain1 = BigCake()  
 self.chain2 = MediumCake()  
 self.chain3 = SmallCake()  
  
 self.chain1.set\_successor(self.chain2)  
 self.chain2.set\_successor(self.chain3)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 CAKES = CakeChain()  
 AMOUNT = int(input(**'Введите желаемую сумму всех тортов: '**))  
 **if** AMOUNT < 1:  
 print(**'Введите положительное число'**)  
 exit()  
 CAKES.chain1.handle(AMOUNT)  
 print(**'Готово!'**)

test\_mock.py

**import** unittest**from** unittest.mock **import** patch  
**import** io  
*# from github.lab4.chain\_of\_responsibility import \****from** chain\_of\_responsibility **import** \*  
  
  
**class** TestFoundation(unittest.TestCase):  
  
 @patch(**'sys.stdout'**, new\_callable=io.StringIO)  
 **def** test\_foo\_one(self, mock\_stdout):  
 CakeChain().chain1.handle(6)  
 **assert** mock\_stdout.getvalue() == **'Больших тортов весом 5 кг -- 1\nМаленьких тортов весом 1 кг -- 1\n'  
  
 def** test\_foo\_two(self):  
 **with** patch(**'sys.stdout'**, new=io.StringIO()) **as** fake\_stdout:  
 CakeChain().chain1.handle(5)  
  
 **assert** fake\_stdout.getvalue() == **'Больших тортов весом 5 кг -- 1\n'  
  
  
if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 unittest.main()

**Экранные формы с примерами выполнения программы:**











