Лабораторная работа № 7

Мой вариант: 10.

3. Напишите класс «Стол», на который можно ставить различные столовые приборы (производные классы от базового класса "Вещь") и убирать их со стола. У каждой вещи должно быть имя и стоимость. Для класса "стол" реализуйте 2 метода поиска вещи по имени и стоимости, а также метод подсчета общей стоимости вещей на столе. На стол можно положить только ограниченное количество приборов. Требуется реализовать возможность вывода текущих состояний объектов в терминал.

```
'''
writed by Bulat Zaripov
'''
from typing import List


class TableIsFull(Exception):
    "Called if the number of objects on the table is greater than it
should be"


class Thing:
    '''
    class thing
    '''
    def __init__(self, name: str, cost: int):
        self.name = name
        self.cost = cost

    def __str__(self):
        return f"{self.name} {self.cost}"


class Toy(Thing):
    '''
    class thing
    '''
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)


class Table:
```

```
    '''
    class table
    '''
    def __init__(self, table_size:int = 10, things: List[Thing] = []):
        self.table_size = table_size
        self.things = things

    def get_table_cost(self):
        return sum(thing.cost for thing in self.things)

    def add_thing(self, thing: Thing):
        if len(self.things) >= self.table_size:
            raise TableIsFull
        self.things.append(thing)

    def delete_thing(self, search_name: str):
        index = self.get_index_by_name(search_name)
        del self.things[index]

    def get_count_things(self):
        return len(self.things)

    def find_by_name(self, search_name: str):
        index = self.get_index_by_name(search_name)
        return self.things[index]

    def get_index_by_name(self, search_name: str):
        for index, thing in enumerate(self.things):
            if search_name == thing.name:
                return index
        raise ValueError(f"Name {search_name} is not found")

    def find_by_cost(self, search_cost: int):
        for thing in self.things:
            if thing.cost == search_cost:
                return thing
        raise ValueError(f"Cost {search_cost} is not found")

    def __str__(self):
        name = "Table:"
        for thing in self.things:
            name += "\n" + str(thing)
        return name
```

Тестирование системы

```
import unittest
```

```
from main import *


class TestTable(unittest.TestCase):
    @classmethod
    def setUpClass(self):
        self.table = Table()

    def test_add_thing_in_table(self):
        self.table.add_thing(Thing('tomat', 20))

        self.assertEqual(len(self.table.things), 1)
        self.assertEqual(self.table.get_table_cost(), 20)

        self.assertEqual(self.table.things[0].name, 'tomat')
        self.assertEqual(self.table.things[0].cost, 20)

    def test_search_on_table(self):
        self.table.add_thing(Thing('tomat', 20))
        tomat_index = self.table.get_index_by_name('tomat')
        self.assertEqual(self.table.things[tomat_index].name, 'tomat')
        self.assertEqual(self.table.find_by_cost(20).name, 'tomat')

        self.assertRaises(ValueError, self.table.find_by_name, 'notomat')
        self.assertRaises(ValueError, self.table.find_by_cost, 30)

    def test_delete_thing(self):
        self.table.add_thing(Thing('thing_to_delete', 1))
        count_of_things_now = self.table.get_count_things()
        self.table.delete_thing('thing_to_delete')
        count_of_things_after = self.table.get_count_things()
        self.assertEqual(count_of_things_now, count_of_things_after+1)

    def test_add_things_more_than_table_size(self):
        for i in range(self.table.get_count_things(),
self.table.table_size):
            self.table.add_thing(Thing(f"thing{i}", i))
        self.assertRaises(TableIsFull, self.table.add_thing, Thing("over",
20))

    def tearDown(self):
        self.table.things = []
```

11. Напишите класс «Товар» и «Заказ». Заказ может состоять из произвольного числа товаров, которые формируют его конечную стоимость. Реализуйте возможность давать скидку на различные типы товаров в заказе и на всю стоимость заказа. Также реализуйте возможность узнавать в заказе стоимость только задаваемого типа

товаров. Требуется реализовать возможность вывода текущих состояний объектов в терминал.

```
from typing import List


class Product:
    def __init__(self, name: str, cost: float, category: str):
        self.name = name
        self.start_cost = cost
        self.category = category
        self.cost = cost


class Promotion:
    '''
    Скидка на всё корзину
    '''
    def __init__(self, name: int, procent: float):
        self.name = name
        self.procent = procent

    def use(self, products: List[Product]) -> List[Product]:
        for product in products:
            product.cost = product.start_cost * self.procent


class CategoryPromotion(Promotion):
    '''
    Скидка на категорию
    '''
    def __init__(self, name: str, procent: float, category: str):
        super().__init__(name, procent)
        self.category = category

    def use(self, products: List[Product]) -> List[Product]:
        for product in products:
            if product.category == self.category:
                product.cost = product.start_cost * self.procent
        return products


class Order:
    '''
    Заказ
    '''
    def __init__(self):
        self.products = []
        self.promotions = []
```

```
        self.cost = 0

    def add_product(self, product: Product):
        self.products.append(product)
        self.calculate()

    def add_promotion(self, promotion: Promotion):
        self.promotions.append(promotion)
        self.calculate()

    def get_cost_by_category(self, category: str) -> int:
        return sum(el.cost for el in self.products if el.category ==
category)

    def calculate(self):
        for promo in self.promotions:
            promo.use(self.products)
        self.cost = sum(product.cost for product in self.products)

    def print_order(self):
        print('Order')
        for promo in self.promotions:
            print('promo', promo.name, "sale", promo.procent)
        for product in self.products:
            print('product', product.name, product.cost, product.category)
```

Тестирование

```
import unittest

from main import *


class TestOrder(unittest.TestCase):
    @classmethod
    def setUpClass(self):
        self.order = Order()

    @classmethod
    def tearDownClass(self):
        del self.order

    def test_add_product(self):
        self.order.add_product(Product('cheese', 100, 'milk'))
        self.assertEqual(len(self.order.products), 1)

    def test_add_promotion(self):
        self.order.add_promotion(Promotion('SummerPromo20', 0.8))
        self.assertEqual(len(self.order.promotions), 1)
```

```
    def test_cost(self):
        self.assertEqual(self.order.cost, 80.0)
```