lecture_18

October 16, 2022

1 Lecture 18

[30]:
```
from abc import ABC, abstractmethod

class Container:
    def __init__(self):
        self.content = []

    @property
    def weight(self):
        return sum(c.weight for c in self.content)

    @abstractmethod
    def add_items(self, *args):
        raise NotImplementedError()
```

[32]:
```
class Parcel(Container):
    def __init__(self, destination_address, max_weight):
        super().__init__()
        self.destination_address = destination_address
        self.max_weight = max_weight

    def add_items(self, *items):
        total_weight = sum(item.weight for item in items)
        if self.weight + total_weight > self.max_weight:
            raise Exception("Can't fit this product into the box")
        self.content.extend(items)
```

[33]:
```
class Box(Container):
    def __init__(self, height, width, length):
        super().__init__()
        self.width = width
        self.length = length
        self.height = height

    @property
    def volume(self):
```

```
        return self.width * self.length * self.height

    def add_items(self, *items):
        self.content.extend(items)
```

[34]:
```
class Product:
    def __init__(self, name, weight):
        self.name = name
        self.weight = weight
```

[35]:
```
phone = Product(name="iPhone 14", weight=700)
```

[36]:
```
headphone = Product(name="iPhone headphone", weight=100)
```

[37]:
```
box_1 = Box(100, 100, 50)
```

[38]:
```
box_1.add_items(phone, headphone)
```

[39]:
```
charger = Product(name="iPhone Charger", weight=200)
```

[40]:
```
box_2 = Box(50, 50, 20)
```

[42]:
```
box_2.add_items(charger)
```

[43]:
```
hammer = Product(name="Hammer", weight=1500)
```

[44]:
```
box_3 = Box(200, 50, 20)
```

[45]:
```
box_3.add_items(hammer)
```

[46]:
```
box_4 = Box(150, 150, 70)
```

[47]:
```
box_4.add_items(box_1, box_2)
```

[48]:
```
box_4.content
```

[48]:
```
[<__main__.Box at 0x104713640>, <__main__.Box at 0x10433d910>]
```

[49]:
```
class Reciept(Product):
    def __init__(self):
        super().__init__(name="Recipt", weight=0)
```

[50]:
```
reciept = Reciept()
```

[51]:
```
parcel_1 = Parcel(destination_address="Baghramyan 26, Yerevan, Armenia",␣
↪max_weight=1500)
```

[52]:
```
parcel_1.add_items(box_3, box_4)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Input In [52], in <cell line: 1>()
----> 1 parcel_1.add_items(box_3, box_4)

Input In [32], in Parcel.add_items(self, *items)
      8 total_weight = sum(item.weight for item in items)
      9 if self.weight + total_weight > self.max_weight:
---> 10     raise Exception("Can't fit this product into the box")
     11 self.content.extend(items)

Exception: Can't fit this product into the box
```

[53]:
```
parcel_1.add_items(box_4)
```

[54]:
```
parcel_1.add_items(box_3)
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Input In [54], in <cell line: 1>()
----> 1 parcel_1.add_items(box_3)

Input In [32], in Parcel.add_items(self, *items)
      8 total_weight = sum(item.weight for item in items)
      9 if self.weight + total_weight > self.max_weight:
---> 10     raise Exception("Can't fit this product into the box")
     11 self.content.extend(items)

Exception: Can't fit this product into the box
```

[58]:
```
parcel_1.content[0].content
```

[58]:
```
[<__main__.Box at 0x104713640>, <__main__.Box at 0x10433d910>]
```

1.1 Observer

[85]:
```
class Product:
    def __init__(self, name):
        self.name = name


class Shop:  # Publisher
    def __init__(self, name):
```

```
        self.name = name
        self.products = []
        self.subscribers = {
            'iphone': set(),
            'ipad': set(),
            'macbook': set(),
        }

    def add_subscriber(self, product_name, subscriber):
        self.subscribers[product_name].add(subscriber)

    def update_subscribers(self, product_name):
        for subscriber in self.subscribers[product_name]:
            subscriber.get_notified(product_name)

    def receive_new_product(self, product):
        self.products.append(product)
        self.update_subscribers(product.name)


class Subscriber(ABC):
    @abstractmethod
    def get_notified(self, event):
        raise NotImplementedError()


class Customer(Subscriber):  # Subscriber
    def __init__(self, name):
        self.name = name

    def get_notified(self, event):
        print("Omw to buy it")
```

[86]:
```
customer_1 = Customer("Adam Smith")
```

[87]:
```
customer_2 = Customer("Jack Adams")
```

[88]:
```
product = Product('iphone')
```

[89]:
```
shop_1 = Shop("iStore")
```

[90]:
```
shop_1.add_subscriber('iphone', customer_1)
shop_1.add_subscriber('iphone', customer_1)
shop_1.add_subscriber('iphone', customer_1)
shop_1.add_subscriber('iphone', customer_1)
shop_1.add_subscriber('ipad', customer_2)
```

[91]:
```
shop_1.receive_new_product(product)
```

```
Omw to buy it
```

[92]:
```
shop_1.subscribers
```

[92]:
```
{'iphone': {<__main__.Customer at 0x105dd2790>},
 'ipad': {<__main__.Customer at 0x105e5ccd0>},
 'macbook': set()}
```

[ ]:
```

```