lecture_17

October 16, 2022

1 Lecture 17

1.1 Factory Pattern

[2]:
```python
from abc import ABC, abstractmethod
```

[40]:
```python
class Creator(ABC):
    @abstractmethod
    def factory_method(self, name):
        raise NotImplementedError

    def show_products(self, name):
        customer = self.factory_method(name)

        products = ['bread', 'milk', 'chocolate']

        if customer.can_buy_alcohol():
            products += ['beer', 'wine']

        return products
```

[41]:
```python
class Customer(ABC):
    def __init__(self, name):
        self.name = name

    @abstractmethod
    def can_buy_alcohol(self):
        return NotImplementedError
```

[42]:
```python
class OrdinaryCustomer(Customer):
    def can_buy_alcohol(self):
        return True


class UnderagedCustomer(Customer):
    def can_buy_alcohol(self):
        return False
```

[53]:
```
class OrdinaryCustomerCreator(Creator):
    def factory_method(self, name):
        return OrdinaryCustomer(name)
```

[54]:
```
class UnderagedCustomerCreator(Creator):
    def factory_method(self, name):
        return UnderagedCustomer(name)
```

[55]:
```
def display_products_to_customer(customer_factory, name):
    print(customer_factory.show_products(name))
```

[56]:
```
ordinary_customer_creator = OrdinaryCustomerCreator()
underaged_customer_creator = UnderagedCustomerCreator()
```

[62]:
```
age = input()
age = int(age) if age != '' else -1
name = input()
```

```
 adam
```

[60]:
```
if age < 18:
    display_products_to_customer(underaged_customer_creator, name)
else:
    display_products_to_customer(ordinary_customer_creator, name)
```

```
['bread', 'milk', 'chocolate']
```

[61]:
```
class UnverifiedCustomer(Customer):
    def can_buy_alcohol(self):
        return False
```

[64]:
```
class UnverifiedCustomerCreator(Creator):
    def factory_method(self, name):
        return OrdinaryCustomer(name)
```

[65]:
```
unverified_customer_creator = UnverifiedCustomerCreator()
```

[67]:
```
age = input()
age = int(age) if age != '' else -1
name = input()
```

```
 Adam
```

[68]:
```
if age == -1:
    display_products_to_customer(unverified_customer_creator, name)
```

```
elif age < 18:
    display_products_to_customer(underaged_customer_creator, name)
else:
    display_products_to_customer(ordinary_customer_creator, name)
```

```
['bread', 'milk', 'chocolate', 'beer', 'wine']
```

1.2 Metaclasses

[69]:
```
class Foo:
    def bar(self):
        print("okay")
```

[70]:
```
foo = Foo()
```

[71]:
```
foo.bar()
```

```
okay
```

[72]:
```
foo()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [72], in <cell line: 1>()
----> 1 foo()

TypeError: 'Foo' object is not callable
```

[73]:
```
foo("test")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [73], in <cell line: 1>()
----> 1 foo("test")

TypeError: 'Foo' object is not callable
```

[74]:
```
class Foo:
    def __call__(self):
        print("It works!")
```

[75]:
```
foo = Foo()
```

[76]:
```
foo()
```

```
It works!
```

```
[80]: class Foo:  # Callable
          def __call__(self, text):
              print("It works!")
              print(text)
```

```
[81]: foo = Foo()
```

```
[82]: foo("test")
```

```
It works!
test
```

```
[83]: Bar = Foo
```

```
[84]: type(Bar)
```

```
[84]: type
```

```
[85]: b = Bar()
```

```
[86]: b
```

```
[86]: <__main__.Foo at 0x107acce80>
```

```
[87]: type(type)
```

```
[87]: type
```

```
[91]: a = Foo()
      b = Foo()
```

```
[92]: a is b
```

```
[92]: False
```

```
[93]: a == b
```

```
[93]: False
```

```
[94]: id(a)
```

```
[94]: 4570504208
```

```
[95]: id(b)
```

```
[95]: 4563945696
```

[99]:
```
class SingletonMeta(type):
    instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls.instances:
            inst = super().__call__(*args, **kwargs)
            cls.instances[cls] = inst
            return inst
        return cls.instances[cls]
```

[100]:
```
class FooSingleton(metaclass=SingletonMeta):
    def bar(self):
        print("okay")
```

[101]:
```
a = FooSingleton()
```

[102]:
```
id(a)
```

[102]:
```
4563429072
```

[103]:
```
b = FooSingleton()
```

[104]:
```
id(b)
```

[104]:
```
4563429072
```

[105]:
```
a is b
```

[105]:
```
True
```

[106]:
```
c = FooSingleton()
```

[107]:
```
c is a
```

[107]:
```
True
```

[108]:
```
c.bar()
```

```
okay
```

[109]:
```
c.t = "test"
```

[110]:
```
c.t
```

[110]:
```
'test'
```

[111]:
```
a.t
```

[111]: 'test'

[112]:
```
b.t
```

[112]: 'test'

[113]:
```
class SingletonMeta(type):
    instance = None

    def __call__(cls, *args, **kwargs):
        if cls.instance is None:
            cls.instance = super().__call__(*args, **kwargs)
        return cls.instance
```

[114]:
```
def Bar(metaclass=SingletonMeta):
    pass
```

[115]:
```
Bar() is Bar()
```

[115]: True

[ ]:
```
```