프로그래밍을 위한 설계

소요시간 : 6 min

> 클래스를 설계하며 기능을 명확히 구분해봅니다.

OOP 개념을 활용하여 코드설계를 해보자

클래스 설계와 사용

- 여기서 주로 배울 것을 클래스 설계이다.
- 클래스 설계가 중요한 이유는 **코드 재사용성**이다.

💡 생각하는 시간 : 아래처럼 단계별로 **코드설계**를 진행한다.

- 코드설계를 진행하는데 앞서, 바로 소스코드를 작성하지 않고 그림과 글을 통해 자신의 방법으로 **코드블록을 구성**해본다.

1단계

- 코드 설계 시 사용할 object부터 적어보자.
- Users
    - Customers
    - Vendors
    - Admin
- Products
- Purchases

2단계

- 코드 작성 전, 각 object별로 요구되는 속성과 어떤 기능을 위해 생성되었는지 설계한다.

        - 이름
        - 사용자가 관리자인지?
    - Customers
        - Attributes
            - 이름
            - 구매목록
    - Vendors
        - Attributes
            - 이름
            - 상품목록
    - Admin
        - 이름
        - 사용자가 관리자임을 나타내는 구분값
- Products
    - Attributes
        - 이름
        - 가격
        - 공급업체
- Purchases
    - Attributes
        - 제품
        - 고객
        - 가격
        - 구매완료기간

object간 관계에 대해 생각해본다.

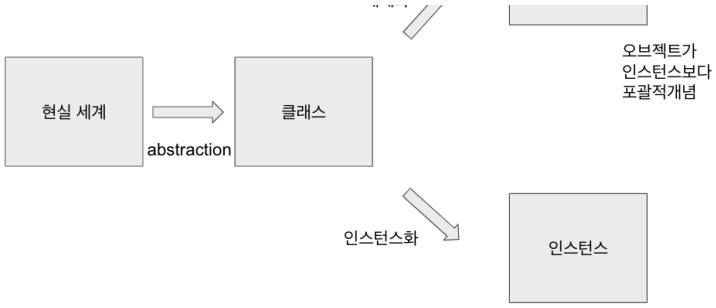
- 판매자는 1개 이상의 제품을 갖고 있다.
- 고객은 1개 이상의 구매를 한다.
- 구매는 1개 이상의 제품을 구매하는 것이다.

```
class User:
    def __init__(self, name, is_admin=False)
        self.name = name
        self.is_admin = is_admin


# User로부터 상속받는 3개의 클래스를 정의

class User:
    def __init__(self, name, is_admin=False)
        self.name = name
        self.is_admin = is_admin

class Admin(User):
    def __init__(self, name):
        super().__init__(name, is_admin=True

class Customer(User):
    def __init__(self, name):
        super().__init__(name)
        self.purchases = []

class Vendor(User):
    def __init__(self, name):
        super().__init__(name)
        self.products = []
```

```
from datetime import datetime

class User:
    def __init__(self, name, is_admin=False)
        self.name = name
        self.is_admin = is_admin

class Admin(User):
    def __init__(self, name):
        super().__init__(name, is_admin=True)

class Customer(User):
    def __init__(self, name):
        super().__init__(name)
        self.purchases = []

class Vendor(User):
    def __init__(self, name):
        super().__init__(name)
        self.products = []

class Product:
    def __init__(self, name, price, vendor):
        self.name = name
        self.price = price
        self.vendor = vendor

class Purchase:
    def __init__(self, product, customer):
        self.product = product
        self.customer = customer
        self.purchase_price = product.price
        self.purchase_data = datetime.now()
```

```
from datetime import datetime

class User:
    def __init__(self, name, is_admin=False)
        self.name = name
        self.is_admin = is_admin

class Admin(User):
    def __init__(self, name):
        super().__init__(name, is_admin=True

class Customer(User):
    def __init__(self, name):
        super().__init__(name)
        self.purchases = []
    def purchase_product(self, product):
        purchase = Purchase(product, self)
        self.purchases.append(purchase)

class Vendor(User):
    def __init__(self, name):
        super().__init__(name)
        self.products = []
    def create_product(self, product_name, pr
        product = Product(product_name, produ
        self.products.append(product)

# 모델링을 위한 추가 소스코드
class Product:
    def __init__(self, name, price, vendor):
        self.name = name
        self.price = price
        self.vendor = vendor

class Purchase:
    def __init__(self, product, customer):
        self.product = product
        self.customer = customer
        self.purchase_price = product.price
        self.purchase_data = datetime.now()
```

클래스의 인스턴스화



- 클래스를 생성했으면, 그것을 활용하기 위한 인스턴스화가 필요하다.
    - object가 생성된 이후, object가 소프트웨어의 **메모리할당이 되면 인스턴스**가 되는 것이다.
        - object는 인스턴스를 포함할 수 있으며, 포괄적 의미를 갖는다.
        - object는 프로그래밍 전체에서 쓰이는 포괄적인 의미를 갖으므로 인스턴스와 비교하면서 학습하는 대상은 아니다.

```
# 가장 기본적인 클래스를 생성해보고 값을 확인해본

class MyFirstClass:
    pass

a = MyFirstClass()  # 인스턴스화(메모리할당됨)

print(a)    # 주소값은 일반적인 정수값과 다르게
```

Mark as Completed

< Prev Next >