

[과제5]클래스메서드와 함수리턴타입 이용한 간단한 책대여서비스 만들기

클래스 메서드는 Python의 클래스 메서드 데코레이터 `@classmethod` 와 함께 정의된 메서드입니다. 간단히 설명드리면:

```
@classmethod
def get_all_books(cls) → QuerySet['Book']:
```

클래스 메서드는 다음과 같은 기능과 구조를 가집니다:

✓ 구성 요소 분석

1. `@classmethod`
 - 이 데코레이터는 메서드가 클래스 자체를 첫 번째 인자로 받도록 합니다. 즉, `cls` 는 클래스 (`Book` 클래스)를 의미합니다.
 - 클래스의 인스턴스를 만들지 않고도 호출할 수 있습니다. 예: `Book.get_all_books()`
2. `def get_all_books(cls)`
 - `cls` 는 호출한 클래스 자체를 가리키며, 해당 클래스에 대한 쿼리 수행 등에 사용됩니다.
3. `> QuerySet['Book']`
 - 이는 Python의 함수 리턴 타입 힌트입니다.
 - 이 메서드는 `Book` 객체들로 구성된 `QuerySet` 을 반환함을 의미합니다.
 - `QuerySet` 은 Django ORM에서 DB 쿼리 결과를 나타내는 자료형입니다.

✓ 실제 예시 (Django 모델에 적용)

예를 들어 `Book` 모델이 있다고 하면:

```
from django.db import models
from typing import TYPE_CHECKING
if TYPE_CHECKING:
```

```
from .models import Book # 타입 힌트 순환 참조 방지용
```

```
class Book(models.Model):  
    title = models.CharField(max_length=200)  
    author = models.CharField(max_length=100)  
  
    @classmethod  
    def get_all_books(cls) → models.QuerySet['Book']:  
        return cls.objects.all()
```

✓ 사용 예시

```
books = Book.get_all_books()  
for book in books:  
    print(book.title)
```

이 함수에 필터 조건이나 정렬도 추가할 수 있습니다. 예를 들어:

```
@classmethod  
def get_available_books(cls) → models.QuerySet['Book']:  
    return cls.objects.filter(is_available=True).order_by('title')
```

`@classmethod`를 활용한 `get_all_books` 메서드를 포함한 간단한 Django 예제. 아래는 **Book** 모델과 함께 책 목록을 조회하는 클래스 메서드를 사용한 구조를 포함합니다.

✓ 목표

- `Book` 모델 정의
- `get_all_books()` 클래스 메서드 구현
- 책 데이터를 DB에 저장
- 클래스 메서드를 호출해서 모든 책 출력

1. Django 프로젝트 기본 구조

```
django-admin startproject mylibrary
cd mylibrary
python manage.py startapp books
```

2. settings.py 설정

mylibrary/settings.py 에 앱 등록:

```
INSTALLED_APPS = [
    ...
    'books',
]
```

3. Book 모델 정의

books/models.py :

```
from django.db import models
from django.db.models import QuerySet

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)

    def __str__(self):
        return f"{self.title} by {self.author}"

    @classmethod
    def get_all_books(cls) → QuerySet['Book']:
        return cls.objects.all()
```

4. 마이그레이션

```
python manage.py makemigrations
python manage.py migrate
```

5. 샘플 데이터 등록 (관리자 없이)

`books/management/commands/seed_books.py` 생성

```
mkdir -p books/management/commands
touch books/management/__init__.py
touch books/management/commands/__init__.py
touch books/management/commands/seed_books.py
```

`books/management/commands/seed_books.py` :

```
from django.core.management.base import BaseCommand
from books.models import Book

class Command(BaseCommand):
    help = 'Insert sample books'

    def handle(self, *args, **kwargs):
        Book.objects.create(title='1984', author='George Orwell')
        Book.objects.create(title='Brave New World', author='Aldous Huxley')
        Book.objects.create(title='Fahrenheit 451', author='Ray Bradbury')
        self.stdout.write(self.style.SUCCESS('Sample books inserted!'))
```

실행:

```
python manage.py seed_books
```

6. 클래스 메서드 사용해보기 (Shell에서)

```
python manage.py shell
```

```
from books.models import Book

books = Book.get_all_books()
```

```
for book in books:
    print(book)
```

출력:

```
1984 by George Orwell
Brave New World by Aldous Huxley
Fahrenheit 451 by Ray Bradbury
```

결과 정리

- `get_all_books()` 는 `Book` 클래스 전체 데이터를 가져오는 클래스 메서드
- 장고 ORM의 `QuerySet` 을 반환
- shell이나 뷰에서 사용 가능

`@classmethod` 와 `QuerySet['Book']` 타입 힌트를 그대로 활용해서 **조건에 따른 책 조회 기능**을 더 만들어 봄

목표

기존 `get_all_books()` 에 더해 아래 기능도 클래스 메서드로 구현:

1. `get_books_by_author(cls, author_name)` → 특정 저자의 책만 조회
2. `get_books_by_title_keyword(cls, keyword)` → 제목에 특정 키워드가 포함된 책 조회
3. `get_books_ordered_by_title(cls)` → 책 제목 순으로 정렬된 목록 조회

전체 코드 (books/models.py)

```
from django.db import models
from django.db.models import QuerySet

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)
```

```

def __str__(self):
    return f"{self.title} by {self.author}"

@classmethod
def get_all_books(cls) → _____: # → 여기에 동작 코드를 작성하세요(1점)
    """
    전체 책 목록 반환
    """
    return _____ # → 여기에 동작 코드를 작성하세요(2점)

@classmethod
def get_books_by_author(cls, _____) → QuerySet['Book']: # →
여기 괄호안에 동작 코드를 작성하세요(1점)
    """
    특정 저자의 책만 반환
    """
    return cls.objects.filter(_____ = author_name) # → 여기 괄호
안에 동작 코드를 작성하세요(2점)

@classmethod
def get_books_by_title_keyword(cls, _____) → QuerySet['Book']: # →
여기 괄호안에 동작 코드를 작성하세요(1점)
    """
    제목에 키워드가 포함된 책 반환 (대소문자 구분 없이)
    """
    return cls.objects.filter(_____ = keyword) # → 여기에 동작 코드를
작성하세요(2점)

@classmethod
def get_books_ordered_by_title(cls) → _____: # → 여기 괄호안에
동작 코드를 작성하세요(1점)
    """
    제목 순으로 정렬된 책 목록 반환
    """
    return cls.objects.all()._____ # → 여기 괄호안에 동작 코드를 작성
하세요(2점)

```

사용 예시 (Django shell)

```
python manage.py shell
```

```
from books.models import Book
```

1. 전체 책 조회

```
all_books = Book.get_all_books()
```

```
print("📖 전체 책 목록:")
```

```
for _____: # → 아래 출력과 같이 출력되도록 shell 스트립트 작성  
    하세요(1점)
```

```
    print(book)
```

2. 특정 저자 책만 조회

```
orwell_books = Book.get_books_by_author("George Orwell") # → 아래  
출력과 같이 출력되도록 shell 스트립트 작성하세요(1점)
```

```
print("\n📖 George Orwell 책 목록:")
```

```
for _____:
```

```
    print(book)
```

3. 제목 키워드로 검색

```
dystopia_books = Book.get_books_by_title_keyword("new") # → 아래 출  
력과 같이 출력되도록 shell 스트립트 작성하세요(1점)
```

```
print("\n🔍 제목에 'new'가 포함된 책 목록:")
```

```
for _____:
```

```
    print(book)
```

4. 제목순 정렬


```
sorted_books = Book.get_books_ordered_by_title() # → 아래 출력과 같이  
출력되도록 shell 스트립트 작성하세요(1점)
```

```
print("\n📖 제목순 정렬:")
```

```
for _____:
```

```
    print(book)
```


출력 예시

 전체 책 목록: #---> 아래 출력을 캡처하여 제출하세요 (1점)


1984 by George Orwell

Brave New World by Aldous Huxley


Fahrenheit 451 by Ray Bradbury

 George Orwell 책 목록: #---> 아래 출력을 캡처하여 제출하세요 (1점)

1984 by George Orwell

 제목에 'new'가 포함된 책 목록: #---> 아래 출력을 캡처하여 제출하세요 (1점)

Brave New World by Aldous Huxley

 제목순 정렬: #---> 아래 출력을 캡처하여 제출하세요 (1점)

1984 by George Orwell

Brave New World by Aldous Huxley

Fahrenheit 451 by Ray Bradbury

`author__iexact` 는 Django ORM (Object-Relational Mapping)에서 **필터링 조건**으로 사용되는 표현입니다. 구체적으로 설명하자면:

◆ 의미

`__iexact` 는 대소문자를 구분하지 않는 정확한 일치 의미를 의미합니다.

즉, `author__iexact="J.K. Rowling"` 이라고 하면:

- `'j.k. rowling'`
- `'J.K. ROWLING'`
- `'J.k. RoWiInG'`

모두 매치됩니다.

◆ 사용 예시

```
from myapp.models import Book
```

```
books = Book.objects.filter(author__iexact="J.K. Rowling")
```


이 코드는 `author` 필드가 "J.K. Rowling"과 대소문자를 무시하고 정확히 일치하는 모든 `Book` 객체를 가져옵니다.

◆ 주의할 점

- `__iexact` 는 **부분 일치가 아닌 '정확한 일치'**입니다. 예: "J.K. Rowling"과 "Rowling"은 일치하지 않음.
- **PostgreSQL**이나 **SQLite**는 기본적으로 대소문자 구분이 느슨하지만, 이 옵션은 모든 DB에서 일관된 동작을 보장합니다.

`title__icontains` 는 Django ORM에서 자주 쓰이는 **쿼리 필터 조건** 중 하나로, 의미는 다음과 같습니다: `__icontains` (부분 일치, 대소문자 구분 없음) 의미

◆ 의미

`__icontains` 는:

대소문자를 구분하지 않고 주어진 문자열이 포함되어 있는지를 검사합니다.

즉, `title__icontains="magic"` 이라고 하면:

- `"Magic School"`
- `"The magic of Python"`
- `"MAGICAL Realism"`

등 **"magic"**이라는 단어가 포함된 모든 제목을 찾습니다. (대소문자 무시)

◆ 사용 예시

```
from myapp.models import Book

books = Book.objects.filter(title__icontains="magic")
```

이 코드는 `title` 필드에 `"magic"` 이 포함된 모든 책들을 찾아줍니다.

◆ 차이점 요약

필터	설명
<code>__exact</code>	정확히 일치 (대소문자 구분)
<code>__iexact</code>	정확히 일치 (대소문자 구분 안 함)
<code>__contains</code>	포함 여부 (대소문자 구분)
<code>__icontains</code>	포함 여부 (대소문자 구분 안 함)

- 제출
 - 가상강의실
 - 노션이나 hwp, 워드 등을 사용해서 제출 바랍니다.
- 일정
 - 과제 5 총점 : 20점
 - 4월 30일 (수) 13:30분까지
 - 늦게 제출은 인정하지 않습니다.