

[웹서버컴퓨팅 과제 9] Django 기반 도서 대출 시스템의 서비스 레이어 분리 및 예외 처리

과제 제목:

 Django 기반 도서 대출 시스템의 서비스 레이어 분리 및 예외 처리 예제

핵심 구성 요소:

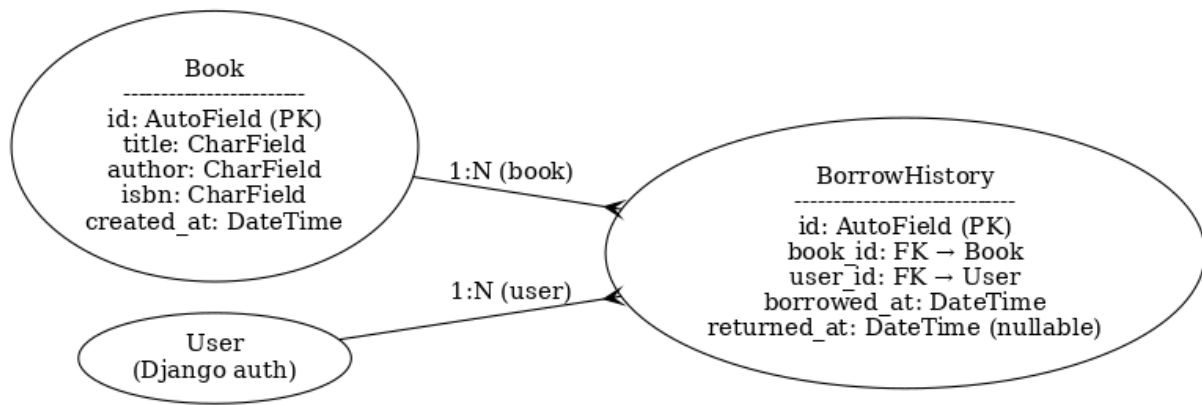
구성	내용
모델	<code>Book</code> , <code>BorrowHistory</code>
서비스 레이어	<code>get_book_by_id()</code> , <code>get_borrow_history_for_book()</code>
사용자 정의 예외	<code>BookNotFound</code> , <code>BookHasNoBorrowHistory</code>
단위 테스트	<code>pytest</code> 를 이용한 서비스 함수 테스트
뷰	<code>book_history()</code> - 예외를 처리하며 템플릿 반환
템플릿	<code>book_history.html</code> , <code>no_history.html</code>

과제 주안점:

- View-Logic 분리 구조
- ORM 함수의 책임 분리 (조회 vs 표현)
- 단위 테스트로 검증 가능한 서비스 구조
- 사용자 정의 예외 처리 설계

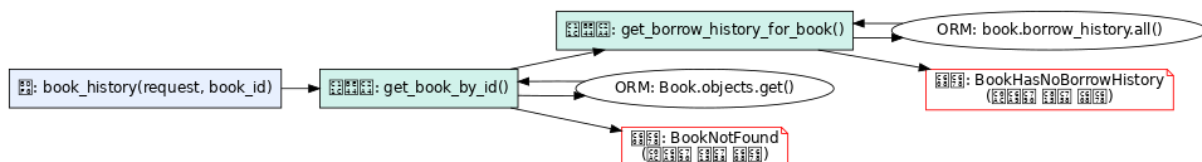
구현할 도서 대출 시스템의 ERD(엔터티 관계도) 시각화 이미지

- `Book` ↔ `BorrowHistory` : 1:N 관계
- `BorrowHistory` ↔ `User` : 1:N 관계 (Django 기본 User 모델 사용)



🔍 구현할 예제의 서비스 흐름도(Service Flow Diagram) 구성 요소:

- **View** : 사용자의 요청 처리 (`book_history`)
- **Service** : `get_book_by_id()` , `get_borrow_history_for_book()` 로직
- **ORM** : Django ORM 함수 호출
- **Exception** : 사용자 정의 예외 (`BookNotFound` , `BookHasNoBorrowHistory`)



1. 클래스 메소드 vs 서비스 레이어 비교

- Django에서의 클래스 메소드(`@classmethod`) 와 서비스 레이어(Service Layer) 구조에서 사용되는 **ORM 함수의 차이점과 역할**. 이 비교를 통해 코드 설계의 방향성과 테스트 전략을 명확하게 이해할 수 있습니다.

✅ 1. 클래스 메소드 vs 서비스 레이어의 ORM 사용 목적

항목	클래스 메소드 (<code>@classmethod</code>)	서비스 레이어 (Service Layer)
✅ 목적	모델 자체의 공통 로직을 캡슐화	뷰 로직을 분리해 비즈니스 로직 담당

✓ 위치	모델 클래스 내부 (<code>models.py</code>)	별도 모듈 (<code>services/*.py</code>)
✓ 호출 방식	<code>Book.recent_books()</code>	<code>book_service.get_recent_books()</code>
✓ 테스트성	단위 테스트 적합	단위 + 통합 테스트 모두 적합
✓ 주요 대상	특정 모델 중심의 정적 쿼리	다양한 모델 간 로직 조합 및 흐름 제어
✓ 재사용 범위	한 모델 내에서 공유	앱 전반에서 로직 공유 가능

✓ 2. ORM 함수 비교

📌 클래스 메소드에서 자주 쓰이는 ORM 함수

```
@classmethod
def available_books(cls):
    return cls.objects.filter(is_available=True)
```

함수	설명
<code>cls.objects.filter()</code>	조건부 객체 목록
<code>cls.objects.get()</code>	단일 객체 조회
<code>cls.objects.create()</code>	새 객체 생성
<code>cls.objects.annotate()</code>	필드 집계

→ 단일 모델에 대해 정적인 쿼리 중심

📌 서비스 레이어에서 자주 쓰이는 ORM 함수

```
def get_books_by_author(author_id):
    author = Author.objects.get(id=author_id)
    return Book.objects.filter(author=author)
```

함수	설명
<code>Model.objects.get()</code>	외래키 조회
<code>Model.objects.select_related()</code>	JOIN 최적화
<code>related_name</code> 으로 역참조 (<code>book.borrow_history.all()</code>)	1:N 연결된 데이터 탐색
<code>exists()</code> , <code>count()</code>	조건 검사 및 수량 확인

→ 다중 모델 간 상호작용, 예외 처리 포함, 흐름 제어 포함

✓ 3. 예제 비교

✓ 클래스 메소드 예 (모델 중심)

```
class Book(models.Model):  
    ...  
  
    @classmethod  
    def recent(cls):  
        return cls.objects.filter(created_at__gte=timezone.now() - timedelta(da  
ys=30))
```

📌 특정 모델(Book) 에 국한된 "최근 등록 도서" 제공

✓ 서비스 함수 예 (로직 조합 + 예외처리)

```
# services/book_service.py  
def get_books_by_keyword_and_user(keyword, user):  
    return Book.objects.filter(title__icontains=keyword, created_by=user).ord  
er_by('-created_at')
```

📌 뷰 사용자의 맥락, 다중 조건, 정렬 포함

✓ 4. 요약 정리

항목	클래스 메소드	서비스 레이어
위치	<code>models.py</code> 내부	<code>services/*.py</code>
관심사	특정 모델의 행위	여러 모델 간의 행위 조합
테스트	단위 테스트 중심	단위 + 통합 테스트
ORM 사용	<code>cls.objects.<ORM 함수>()</code>	<code>Model.objects.<ORM 함수>()</code>
유연성	낮음 (고정된 모델 로직)	높음 (상황 맞춤형 로직)
예외처리	보통 없음	명시적 예외 처리 (<code>try-except</code> , 커스텀 Exception)
실무 사용	공통 조회/필터용 메소드	비즈니스 로직의 표준화된 처리 방식

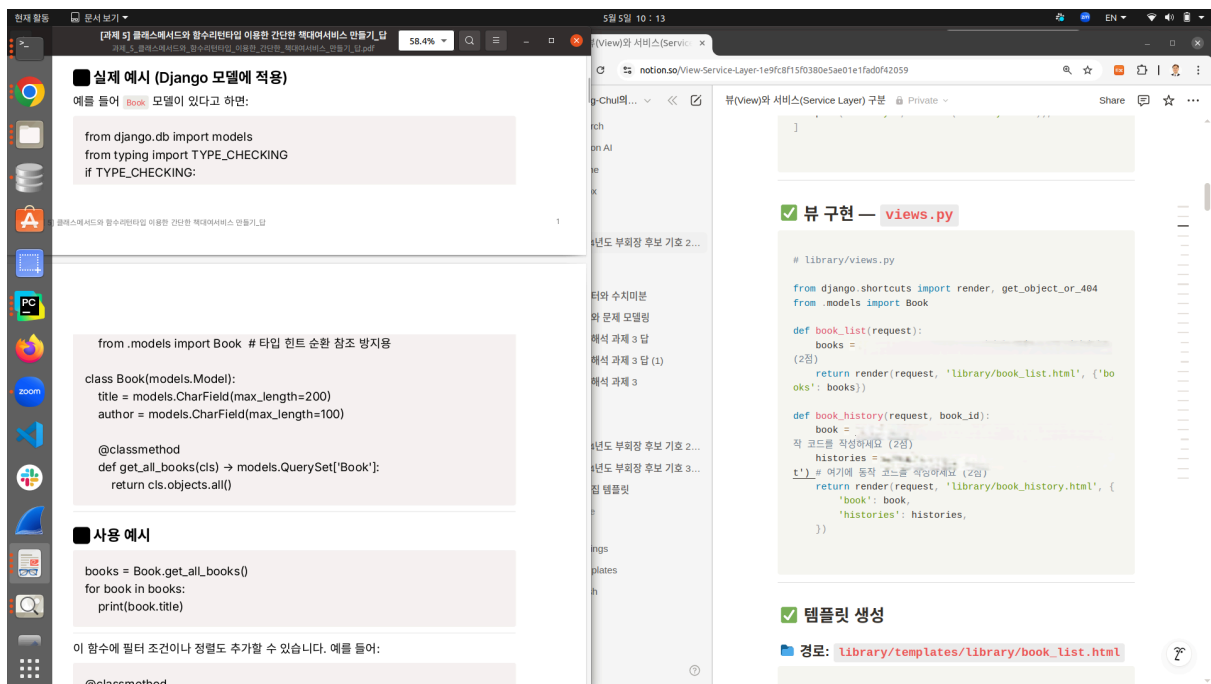
✓ 핵심 요약

💡 ORM 함수는 Django 로직의 엔진이며,
그를 어디서 어떻게 감싸느냐에 따라 "재사용성", "테스트성", "분리
도" 가 결정.

- 클래스메소드: 모델에 직접 붙는 간단한 쿼리 캡슐화
- 서비스레이어: 다양한 ORM 조합과 흐름 제어 중심

2. 클래스 메소드 [이전 과제 5] vs 서비스 레이어의 구분 (이전 과제 5와 비교)

- 클래스메소드와 뷰(View)와 서비스(Service Layer) 구분



- 모델 클래스에 적용된 클래스메소드
 - 클래스메소드에 적용된 ORM 함수

```
cls.objects.all()
```

```
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()
```

- view와 분리된 service layer에 적용된 View와 service layer 구분
 - View와 service layer 구분에 적용된 ORM 함수
 - 그만큼 ORM 함수가 중요한 역할을 하고 있음

```
Book.objects.all()
Book.objects.get()
```

```
def get_book_by_id(book_id):
    try:
        return Book.objects.get(id=book_id) # ✅ ORM 함수
    except Book.DoesNotExist:
        raise BookNotFound()
```

3. 뷰(View)와 서비스(Service Layer) 분리에 사용된 ORM

Django에서 **View와 Service Layer**를 분리하면 유지보수성과 테스트성이 매우 좋아지며, 이 구조에서 핵심이 되는 건 바로 **ORM 함수들**.

아래에서 이 구조를 명확하게 정리하고, **ORM 함수가 각각 어디서 사용되는지**, 왜 중요한지를 설명.

✅ 1. 구조 요약: View vs Service Layer

계층	역할	포함 내용
----	----	-------

View	요청/응답 처리 (입출력)	요청 파싱, 서비스 호출, 템플릿 렌더링
Service Layer	비즈니스 로직 처리	ORM 호출, 예외 처리, 데이터 가공 등

✓ 2. 예제: **book_history** 뷰 분리 구조

views.py (입출력 전담)

```
from django.shortcuts import render
from django.http import HttpResponseNotFound
from library.services import book_service
from library.exceptions import BookNotFound


def book_history(request, book_id):
    try:
        book = book_service.get_book_by_id(book_id)
        histories = book_service.get_borrow_history_for_book(book)
    except BookNotFound:
        return HttpResponseNotFound("책이 존재하지 않습니다.")

    return render(request, 'library/book_history.html', {
        'book': book,
        'histories': histories,
    })
```

services/book_service.py (ORM 처리 집중)

```
from library.models import Book
from library.exceptions import BookNotFound

def get_book_by_id(book_id):
    try:
        return Book.objects.get(id=book_id) # ✓ ORM 함수
    except Book.DoesNotExist:
        raise BookNotFound()
```

```
def get_borrow_history_for_book(book):
    return book.borrow_history.order_by('-borrowed_at') #  역참조 + ORM
```

3. Service Layer에서 자주 쓰이는 ORM 함수

ORM 함수	기능	위치
<code>objects.get()</code>	단일 객체 조회	서비스
<code>objects.filter()</code>	조건 검색	서비스
<code>objects.create()</code>	새 객체 저장	서비스
<code>objects.update()</code>	일괄 수정	서비스
<code>book.borrow_history.all()</code>	1:N 역참조	서비스
<code>exists()</code>	존재 여부 검사	서비스
<code>count()</code>	객체 수 계산	서비스
<code>order_by()</code>	정렬	서비스
<code>annotate()</code>	집계 필드 추가	서비스

4. 왜 ORM 함수가 중요한가?

- 서비스 계층이 하는 일의 대부분이 ORM 쿼리다.
 - 실제 데이터를 생성, 조회, 수정, 삭제하는 책임
- 서비스 로직은 ORM 함수의 조합이다.
 - `get()`, `filter()`, `annotate()` 등은 단순함에도 강력한 조합 도구
- 뷰는 입출력만, ORM은 서비스에서 독립 실행 가능
 - 테스트도 서비스 함수만 호출해서 가능

5. 정리: View와 Service Layer 구분 + ORM 역할

항목	설명
View	사용자 요청을 받아 서비스 호출 후 결과 렌더링
Service	ORM을 이용해 DB에서 데이터 조회/처리/검증
ORM 함수	<code>get</code> , <code>filter</code> , <code>create</code> , <code>order_by</code> , <code>exists</code> 등
중요성	실제 비즈니스 로직의 80%는 ORM 함수로 구현됨

✓ 시각화 예시

```
[View]
↓ 요청 파싱
↓
[Service Layer]
→ ORM: Book.objects.get(), BorrowHistory.objects.filter()
↓
결과 반환
↓
[View] → 템플릿 렌더링 or 응답 반환
```

4. 가장 자주 쓰는 Django ORM 명령어

Django ORM은 데이터베이스 작업을 파이썬 코드로 할 수 있게 해주는 강력한 도구입니다.

아래에 가장 자주 쓰는 Django ORM 명령어들

📌 1. 기본 조회 명령어

명령어	설명
<code>Model.objects.all()</code>	전체 데이터 조회 (<code>QuerySet</code>)
<code>Model.objects.first()</code>	첫 번째 객체
<code>Model.objects.last()</code>	마지막 객체
<code>Model.objects.get(id=1)</code>	조건에 맞는 단일 객체 (없거나 2개 이상이면 예외)
<code>Model.objects.filter(name='홍길동')</code>	조건에 맞는 여러 객체 (QuerySet)
<code>Model.objects.exclude(status='deleted')</code>	조건을 제외한 객체들
<code>Model.objects.order_by('created_at')</code>	정렬 (<code>ASC</code>)
<code>Model.objects.order_by('-created_at')</code>	정렬 (<code>DESC</code>)

📌 2. 조건 & 슬라이싱

명령어	설명
<code>filter(age__gte=20)</code>	나이 20세 이상 (greater than or equal)
<code>filter(name__icontains='kim')</code>	이름에 'kim' 포함 (대소문자 무시)

<code>filter(created_at__year=2024)</code>	특정 연도에 생성된 데이터
<code>filter(status__in=['open', 'closed'])</code>	리스트 안의 값들
<code>[:10]</code>	상위 10개만 가져오기 (슬라이싱)

3. 생성 & 수정 & 삭제

명령어	설명
<code>Model.objects.create(title='test')</code>	새 객체 생성 및 저장
<code>obj = Model(title='test'); obj.save()</code>	생성 후 수동 저장
<code>obj.title = '변경'; obj.save()</code>	필드 값 수정 후 저장
<code>obj.delete()</code>	해당 객체 삭제
<code>Model.objects.filter(...).delete()</code>	여러 개 삭제 (주의!)

4. 관계(Relation) 처리

ForeignKey (1:N)

```
# 예: Comment → Post (ForeignKey(post))
post = Post.objects.get(id=1)
comments = post.comment_set.all() # related_name 안 쓴 경우
# 또는: post.comments.all() (related_name='comments' 사용 시)
```

ManyToMany

```
# 예: Post ↔ Tag
post.tags.add(tag)
post.tags.remove(tag)
post.tags.clear()
```

5. 존재 여부 & 개수

명령어	설명
<code>.exists()</code>	결과가 하나라도 있으면 <code>True</code>
<code>.count()</code>	개수 반환

<code>.distinct()</code>	중복 제거
<code>.values('field')</code>	dict 목록으로 반환 (쿼리 최적화용)
<code>.values_list('id', flat=True)</code>	튜플 or flat list로 반환

6. 고급 기능

명령어	설명
<code>get_or_create()</code>	없으면 생성, 있으면 가져옴
<code>update_or_create()</code>	조건 맞는 거 있으면 업데이트, 없으면 생성
<code>bulk_create([...])</code>	대량 삽입
<code>annotate()</code>	집계 컬럼 추가
<code>aggregate()</code>	집계함수 (Sum, Count 등)
<code>select_related()</code>	ForeignKey 관계 미리 join
<code>prefetch_related()</code>	ManyToMany 등 다대다/역참조 최적화

예제: 게시판 댓글 불러오기

```
post = Post.objects.get(id=1)
for comment in post.comment_set.all():
    print(comment.content)
```

또는 `related_name='comments'` 라면:

```
for comment in post.comments.all():
    print(comment.content)
```

참고용: 단일 객체 vs QuerySet 차이

메서드	반환값	예외 여부
<code>get()</code>	단일 객체	! 없거나 2개 이상이면 예외
<code>filter()</code>	QuerySet	항상 안전

정리 요약

동작	명령어 예시
전체 조회	<code>Model.objects.all()</code>
조건 조회	<code>Model.objects.filter(...)</code>
단일 조회	<code>Model.objects.get(...)</code>
생성	<code>Model.objects.create(...)</code>
수정	<code>obj.save()</code>
삭제	<code>obj.delete()</code>
정렬	<code>order_by('-created_at')</code>
관계 접근	<code>obj.relatedmodel_set.all()</code> or <code>related_name.all()</code>

5. 뷰(View) 와 서비스(Service Layer) 분리 이점

Django에서 뷰(View) 와 서비스(Service Layer) 를 구분하면 다음과 같은 이점이 있음:

- 로직 재사용성 증가
- 테스트 용이
- 뷰는 입출력만 담당 → 가독성 향상

아래에 `library` 앱을 기반으로 한 "뷰/서비스 분리 예제" .

✓ 1. 서비스 디렉토리 구조

```
library/
├── services/
│   └── book_service.py
└── views.py
```

✓ 2. 서비스 코드 (`services/book_service.py`) : 총 3점

```
# library/services/book_service.py

from library.models import Book, BorrowHistory
from django.shortcuts import get_object_or_404

def get_all_books():
```

```

return #_____# # 여기에 동작 코드를 작성하세요 (1점)

def get_book_by_id(book_id: int) → Book:
    return #_____# # 여기에 동작 코드를 작성하세요 (1점)

def get_borrow_history_for_book(book: Book):
    return #_____# # 여기에 동작 코드를 작성하세요 (1점)

```

✓ 3. 뷰 코드 (**views.py**) — 서비스 호출만 수행 : 총 3점

```

# library/views.py

from django.shortcuts import render
from library.services import book_service

def book_list(request):
    books = #_____# # 여기에 동작 코드를 작성하세요 (1점)
    return render(request, 'library/book_list.html', {'books': books})

def book_history(request, book_id):
    book = #_____# # 여기에 동작 코드를 작성하세요 (1점)
    histories = #_____# # 여기에 동작 코드를 작성하세요 (1점)
    return render(request, 'library/book_history.html', {
        'book': book,
        'histories': histories,
    })

```

✓ 4. import 설정 (**library/services/__init__.py**)

```

# library/services/__init__.py
from .book_service import get_all_books, get_book_by_id, get_borrow_history_for_book

```

이렇게 하면 views.py에서는 다음처럼 간단하게 import 가능:

```
from library.services import book_service
```

✓ 정리

계층	역할	위치
View	입출력 처리, 렌더링	<code>views.py</code>
Service	비즈니스 로직, 쿼리 재사용	<code>services/book_service.py</code>
Model	데이터 구조	<code>models.py</code>

6. 뷰(View) 와 서비스(Service Layer) 분리 예제

- 이 예제에서는 `Book(1)` ↔ `BorrowHistory(N)` 구조를 중심으로 모델, URL, 뷰, 템플릿까지 구현
 - `library` 앱을 기반으로 한 1:N 예제를 통해 뷰(View) 와 서비스(Service Layer) 구분함.

✓ 모델 정의 (1:N 구조) — `models.py`

```
# library/models.py

from django.db import models
from django.contrib.auth.models import User

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=100)
    isbn = models.CharField(max_length=20, unique=True)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title

class BorrowHistory(models.Model):
    book = models.ForeignKey(Book, on_delete=models.CASCADE, related_name='borrow_history')
```

```

user = models.ForeignKey(User, on_delete=models.CASCADE)
borrowed_at = models.DateTimeField(auto_now_add=True)
returned_at = models.DateTimeField(null=True, blank=True)

def __str__(self):
    return f"{self.user.username} - {self.book.title}"

```

URL 설정 — `library/urls.py` : 총 3점

```

# library/urls.py

from django.urls import path
from . import views

app_name = 'library'

urlpatterns = [
    path('books/', #_____#, name='book_list'), # 여기에 동작 코드를 작성하세요 (1점)
    path('books/<int:book_id>/history/', #_____#, name='book_history'), # 여기에 동작 코드를 작성하세요 (1점)
]

```

그리고 `config/urls.py` 에도 포함시켜야 합니다:

```

# config/urls.py

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('library/', include('#_____#')), # 여기에 동작 코드를 작성하세요 (1점)
]

```

✓ 뷰 구현 — views.py : 총 3점

```
# library/views.py

from django.shortcuts import render, get_object_or_404
from .models import Book

def book_list(request):
    books = # _____ # # 여기에 동작 코드를 작성하세요 (1점)
    return render(request, 'library/book_list.html', {'books': books})

def book_history(request, book_id):
    book = # _____ # # 여기에 동작 코드를 작성하세요 (1점)
    histories = # _____ # # 여기에 동작 코드를 작성하세요 (1점)
    return render(request, 'library/book_history.html', {
        'book': book,
        'histories': histories,
    })
```

✓ 템플릿 생성 : 총 2점

📁 경로: library/templates/library/book_list.html

```
<h1>📖 책 목록</h1>
<ul>
    {% for book in books %}
        <li>
            <a href="{% url '#_____#' book.id %}"> # 여기에 동작 코드를 작성
하세요 (1점)
                {{ book.title }} by {{ book.author }}
            </a>
        </li>
    {% empty %}
        <li>등록된 책이 없습니다.</li>
    {% endfor %}
</ul>
```


 **경로:** `library/templates/library/book_history.html`

```
<h2><img alt="book icon" data-bbox="191 134 218 151"> 대출 이력: {{ book.title }}</h2>
<ul>
  {% for history in histories %}
    <li>
      {{ history.user.username }} - {{ history.borrowed_at|date:"Y-m-d H:i" }}
      {% if history.returned_at %}
        (반납: {{ history.returned_at|date:"Y-m-d" }})
      {% else %}
        <img alt="clock icon" data-bbox="188 304 215 321"> 미반납
      {% endif %}
    </li>
  {% empty %}
    <li>대출 이력이 없습니다.</li>
  {% endfor %}
</ul>
<a href="{% url '#_____#' %}"><img alt="left arrow icon" data-bbox="141 454 168 471"> 책 목록으로</a> # 여기에 동작 코드를 작성하세요 (1점)
```

관리자 등록 — `admin.py`

```
from django.contrib import admin
from .models import Book, BorrowHistory

admin.site.register(Book)
admin.site.register(BorrowHistory)
```

마이그레이션 및 서버 실행

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

결과 확인

URL	기능
<code>/library/books/</code>	전체 책 목록 보기
<code>/library/books/1/history/</code>	특정 책의 대출 이력 확인

✓ 요약

구성 요소	구현
1:N 구조	Book ↔ BorrowHistory (<code>ForeignKey</code>)
관련 이름	<code>book.borrow_history.all()</code>
URL	<code>library:book_list</code> , <code>library:book_history</code>
페이지 흐름	책 목록 → 각 책의 대출 이력 확인 가능

7. 책(Book) 데이터 자동 생성

책(Book) 데이터를 자동으로 여러 개 생성하는 코드

✓ 1. `Book` 모델 예시 (선행 조건)

```
# library/models.py

from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=100)
    author = models.CharField(max_length=100)
    isbn = models.CharField(max_length=20, unique=True)
    created_at = models.DateTimeField(auto_now_add=True)
```

✓ 2. Django Shell에서 책 여러 개 생성

```
python manage.py shell
```


```
from library.models import Book
from faker import Faker
```

```
import random

fake = Faker()

for i in range(30):
    title = f"{fake.word().title()} Programming {i+1}"
    author = fake.name()
    isbn = f"{fake.random_number(digits=13)}"

    Book.objects.create(
        title=title,
        author=author,
        isbn=isbn
    )
```

 **Faker** 는 가짜 데이터 생성용 라이브러리입니다. 없으면 먼저 설치하세요:

```
pip install Faker
```

✓ 3. 관리 명령어로 자동화하고 싶다면 _ 본 예제에서 사용함

 **library/management/commands/seed_books.py** 생성

```
from django.core.management.base import BaseCommand
from library.models import Book
from faker import Faker

class Command(BaseCommand):
    help = 'Generate random books'

    def handle(self, *args, **kwargs):
        fake = Faker()
        for i in range(30):
            Book.objects.create(
                title=f"{fake.word().title()} Programming {i+1}",
                author=fake.name(),
                isbn=f"{fake.random_number(digits=13)}"
```

```
)  
self.stdout.write(self.style.SUCCESS("✅ 30개의 책 생성 완료!"))
```

폴더 구조

```
library/  
├── management/  
│   └── commands/  
│       └── seed_books.py
```

실행

```
python manage.py seed_books
```

✅ 4. 책 생성 확인

- 관리자 페이지(<http://localhost:8000/admin>) 또는
- 직접 확인:

```
Book.objects.count()  
Book.objects.order_by('-created_at').first()
```

✅ 정리

방법	설명
Shell	빠르게 몇 개 생성
Faker + 반복문	무작위 책 데이터 자동 생성
management command	스크립트로 등록하여 반복 사용 가능

8. 대출 이력(BorrowHistory) 자동 생성

Django에서 **대출 이력(BorrowHistory)** 자동 생성을 위한 관리 명령어(custom management command)

이 명령어를 실행하면 Faker 데이터를 기반으로 랜덤한 대출 이력을 만들어줍니다.

✓ 1. 폴더 구조 생성

아래와 같이 디렉토리 구조를 생성하세요 (없으면 수동으로 만들어야 합니다):

```
library/  
├── management/  
│   ├── commands/  
│   └── seed_borrow.py
```

모든 폴더에 `__init__.py` 파일이 있어야 합니다!

```
touch library/management/__init__.py  
touch library/management/commands/__init__.py
```

✓ 2. `seed_borrow.py` 코드 작성

```
# library/management/commands/seed_borrow.py  
  
from django.core.management.base import BaseCommand  
from library.models import Book, BorrowHistory  
from django.contrib.auth.models import User  
from faker import Faker  
import random  
from datetime import timedelta  
from django.utils import timezone  
  
class Command(BaseCommand):  
    help = '랜덤 대출 이력을 생성합니다.'  
  
    def add_arguments(self, parser):  
        parser.add_argument('--total', type=int, default=50, help='생성할 대출 기록 수')  
  
    def handle(self, *args, **options):  
        fake = Faker()  
        books = list(Book.objects.all())  
        users = list(User.objects.all())  
        total = options['total']
```

```

if not books or not users:
    self.stdout.write(self.style.ERROR("책과 사용자 데이터가 필요합니다.))
    return

for _ in range(total):
    book = random.choice(books)
    user = random.choice(users)
    borrowed_at = fake.date_time_between(start_date='-1y', end_date
    ='now', tzinfo=timezone.utc)
    returned = random.choice([True, False])
    returned_at = borrowed_at + timedelta(days=random.randint(1, 30))
    if returned else None

    BorrowHistory.objects.create(
        book=book,
        user=user,
        borrowed_at=borrowed_at,
        returned_at=returned_at
    )

self.stdout.write(self.style.SUCCESS(f"✅ 대출 기록 {total}개 생성 완료!"))

```

✅ 3. 명령어 실행 방법

```

python manage.py seed_borrow # 기본 50개 생성

# 또는 원하는 개수 지정
python manage.py seed_borrow --total 100

```

✅ 4. 마무리 확인

- Admin에서 확인: <http://localhost:8000/admin/library/borrowhistory/>
- 또는 템플릿 페이지: `/library/books/<id>/history/` 에서 각 책의 대출 이력 확인

✓ 정리

항목	설명
목적	랜덤 대출 이력 자동 생성
명령어	<code>python manage.py seed_borrow --total N</code>
의존 조건	Book, User 데이터가 미리 있어야 함
활용	테스트 데이터, 데모용 샘플 생성 등

9. 예외 처리 포함 서비스 함수

아래는 `services/book_service.py` 파일에 실제로 적용 가능한 **예외 처리 포함 서비스 함수 예제**. 이 예제에서는 `Book` 과 `BorrowHistory` 모델을 기준으로 다음과 같은 예외 상황을 처리합니다:

✓ 예외 상황 정의

함수명	처리할 예외 상황	발생 시점
<code>get_book_by_id()</code>	존재하지 않는 책 ID	<code>Book.DoesNotExist</code>
<code>get_borrow_history_for_book()</code>	대출 이력이 없음	<code>QuerySet.exists() == False</code>

✓ 사용자 정의 예외 클래스 (`exceptions.py`)

```
# library/exceptions.py

class BookNotFound(Exception):
    """요청한 책이 존재하지 않을 경우"""
    pass

class BookHasNoBorrowHistory(Exception):
    """책에 대출 이력이 없을 경우"""
    pass
```

✓ 서비스 레이어 예제 (`services/book_service.py`) : 총 4점

```
# services/book_service.py

from library.models import Book
from library.exceptions import BookNotFound, BookHasNoBorrowHistory

def get_book_by_id(book_id: int) → Book:
    try:
        return Book.objects.get(id=book_id)
    except # _____ #:          # 여기에 동작 코드를 작성하세요 (1점)
        raise # _____ #("ID {book_id}에 해당하는 책이 없습니다.") # 여기에
        동작 코드를 작성하세요 (1점)

def get_borrow_history_for_book(book: Book):
    histories = book.borrow_history.order_by('-borrowed_at')
    if not # _____ #:    # 여기에 동작 코드를 작성하세요 (1점)
        raise # _____ #("{book.title}" 도서에는 대출 이력이 없습니다.") #
        여기에 동작 코드를 작성하세요 (1점)
    return histories
```

✅ 뷰 예제 (**views.py**)에서의 예외 처리 : 총 2점

```
# views.py

from django.shortcuts import render
from django.http import HttpResponseNotFound
from library.services import book_service
from library.exceptions import BookNotFound, BookHasNoBorrowHistory

def book_history(request, book_id):
    try:
        book = book_service.get_book_by_id(book_id)
        histories = book_service.get_borrow_history_for_book(book)
    except # _____ # as e:          # 여기에 동작 코드를 작성하세요 (1점)
        return HttpResponseNotFound(str(e))
    except # _____ # as e: # 여기에 동작 코드를 작성하세요 (1점)
        return render(request, 'library/no_history.html', {'message': str(e)})
```



```
return render(request, 'library/book_history.html', {
    'book': book,
    'histories': histories,
})
```

✓ 장점

이점	설명
✓ 서비스와 뷰의 역할 분리	예외 판단은 서비스에서, 표현은 뷰에서
✓ 단위 테스트 용이	예외 발생 여부를 쉽게 검증 가능
✓ 재사용성 향상	같은 서비스 로직을 다양한 뷰/API에서도 재사용 가능

10. Django에서의 예외 클래스(Exception Class)

Django에서의 **예외 클래스(Exception Class)** 는 Django ORM이나 요청 처리 중 발생하는 **오류를 세밀하게 제어**하고,

사용자에게 적절한 메시지를 제공하거나 백엔드 로직을 안전하게 보호하는 데 핵심적인 역할.

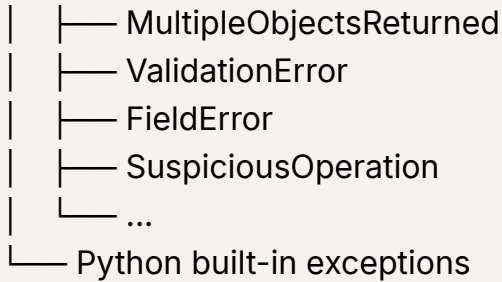
아래에 Django의 **예외 클래스 구조, 계층, 실무 사용 예, 사용자 정의 예외**까지 정리.

✓ 1. 예외 클래스란?

- **예외(Exception)** 는 프로그램 실행 중 **에러 발생 상황**을 명시적으로 처리할 수 있게 해 주는 구조.
- Django는 자체적으로 다양한 예외 클래스를 제공하며, 개발자가 직접 **사용자 정의 예외(custom exception)** 도 만들 수 있음.

✓ 2. 예외 클래스 계층 구조

```
Exception
├── Django-specific exceptions
│   ├── ObjectDoesNotExist
│   └── Book.DoesNotExist ← 모델별 자동 생성
```



✓ 3. 대표적인 Django 예외 클래스

예외 클래스	발생 상황	설명
<code>ObjectDoesNotExist</code>	<code>get()</code> 대상 없음	모든 모델에서 자동 생성되는 <code>.DoesNotExist</code> 가 이를 상속
<code>MultipleObjectsReturned</code>	<code>get()</code> 결과가 여러 개	<code>.get()</code> 은 반드시 하나여야 하는데 2개 이상일 때 발생
<code>ValidationError</code>	<code>full_clean()</code> 또는 Form 검증 실패	모델 필드, 폼 검증 등에서 발생
<code>IntegrityError</code>	DB 무결성 제약 위반	예: Unique 키 중복, NULL 제약 위반 등
<code>DoesNotExist</code>	<code>Model.objects.get()</code> 실패 시	자동 생성 예외 (<code>Book.DoesNotExist</code>)
<code>PermissionDenied</code>	권한 없음	403 에러 발생용
<code>Http404</code>	수동 404 발생	<code>raise Http404("찾을 수 없음")</code>

✓ 4. 예: 모델별 자동 생성 예외

```

from library.models import Book

try:
    book = Book.objects.get(id=999)
except Book.DoesNotExist:
    print("존재하지 않는 책입니다.")
  
```

- `Book.DoesNotExist` 는 `ObjectDoesNotExist` 를 상속한 모델 전용 예외 클래스

✓ 5. 사용자 정의 예외 클래스

직접 만든 서비스 계층에서 명확한 예외 상황을 표현할 수 있음.

```
# library/exceptions.py

class BookNotFound(Exception):
    pass

class BookHasNoBorrowHistory(Exception):
    pass
```

서비스에서 사용:

```
def get_book_by_id(book_id):
    try:
        return Book.objects.get(id=book_id)
    except Book.DoesNotExist:
        raise BookNotFound(f"책 ID {book_id}를 찾을 수 없습니다.")
```

✓ 6. 예외 처리 패턴 요약

```
try:
    # ORM 동작
    book = Book.objects.get(id=10)
except Book.DoesNotExist:
    # 모델 자동 예외
    ...
except ValidationError as e:
    # 폼/모델 검증 오류
    ...
except Exception as e:
    # 일반 예외 (catch-all)
    ...
```

✓ 7. 실무 팁

상황	예외 처리
데이터가 없을 수 있음	<code>.get()</code> → <code>DoesNotExist</code> 예외 처리

사용자 입력 검증 실패	<code>ValidationError</code>
논리상 예외 상황 표현	사용자 정의 예외 사용
API 응답 시	예외 → <code>JsonResponse</code> or <code>HttpResponse(status=404)</code>

✓ 요약

예외 클래스	역할
<code>Book.DoesNotExist</code>	ORM 자동 생성 예외 (객체 없음)
<code>ValidationError</code>	입력값 검증 실패
<code>IntegrityError</code>	DB 무결성 오류
사용자 정의 예외	도메인 상황 명확화 (ex. <code>BookNotFound</code>)

11. 사용자 정의 예외

✓ 정의

```
class BookNotFound(Exception):
    pass
```

- 이 코드는 `Exception` 을 상속하여 새로운 예외 클래스인 `BookNotFound` 를 정의한 것입니다.
- Django의 내장 예외(`DoesNotExist` 등) 외에, 서비스 레이어에서 의미 있는 에러 상황을 명확하게 표현하기 위해 사용합니다.

✓ 왜 사용자 정의 예외를 쓰는가?

목적	설명
의미 명확화	<code>BookNotFound</code> 라는 이름을 통해 어떤 상황인지 명확히 표현
예외 처리 구분	<code>except BookNotFound:</code> 와 같이 구체적으로 분기 가능
코드 가독성 향상	<code>Book.DoesNotExist</code> 보다 도메인 중심 표현이 직관적
서비스 계층 중심 설계	ORM 레벨 예외 → 도메인 예외로 감싸서 노출 최소화

✓ 사용 예시

서비스 함수에서 사용

```
from library.models import Book

class BookNotFound(Exception):
    pass

def get_book_by_id(book_id: int):
    try:
        return Book.objects.get(id=book_id)
    except Book.DoesNotExist:
        raise BookNotFound(f"책 ID {book_id}에 해당하는 도서를 찾을 수 없습니다.")
```

뷰에서 처리

```
from django.http import HttpResponseNotFound
from library.services import book_service
from library.exceptions import BookNotFound

def book_detail_view(request, book_id):
    try:
        book = book_service.get_book_by_id(book_id)
    except BookNotFound as e:
        return HttpResponseNotFound(str(e))

    return render(request, 'library/book_detail.html', {'book': book})
```

정리

항목	설명
<code>BookNotFound</code>	개발자가 정의한 예외 클래스
기반	<code>Exception</code> 을 상속받음
사용 이유	비즈니스 로직에서 발생하는 도메인 에러를 명확히 표현
실사용 위치	서비스 레이어에서 <code>Book.DoesNotExist</code> 를 포장하거나 변환할 때

12. 단위 테스트

단위 테스트도 가능

서비스 단위로 분리해 놓으면 다음처럼 테스트 코드 작성이 용이함:

```
def test_get_book_by_id_success():
    book = Book.objects.create(title='테스트책', author='홍길동', isbn='1234567890123')
    result = get_book_by_id(book.id)
    assert result == book
```

Django에서 서비스(service layer)를 분리했을 때는 뷰를 거치지 않고, 서비스 함수 자체만 테스트하는 단위 테스트(unit test)를 작성할 수 있어 테스트가 훨씬 깔끔하고 빠름.

library/services/book_service.py 서비스 단위 테스트 코드.

예제 서비스 함수 (복습)

```
# library/services/book_service.py

from library.models import Book
from library.exceptions import BookNotFound

def get_book_by_id(book_id: int):
    try:
        return Book.objects.get(id=book_id)
    except Book.DoesNotExist:
        raise BookNotFound(f"ID {book_id}에 해당하는 책이 없습니다.")
```

테스트 디렉토리 구조

```
library/
|— services/
|   └— book_service.py
```

```
└─ tests/
   └─ test_services.py ← 여기서 테스트 작성
```

✓ 테스트 코드 예시: `tests/test_services.py` : 총 6점

```
# library/tests/test_services.py

import pytest
from library.models import Book
from library.services.book_service import get_book_by_id
from library.exceptions import BookNotFound

@pytest.mark.django_db
def test_get_book_by_id_success():
    # Given
    book = Book.objects.#(title='Test Book', author='Tester', isbn='1234567890123') # 여기에 동작 코드를 작성하세요 (1점)

    # When
    result = #(book.id) # 여기에 동작 코드를 작성하세요 (1점)

    # Then
    assert result == # # 여기에 동작 코드를 작성하세요 (1점)
    assert result.# == 'Test Book' # 여기에 동작 코드를 작성하세요 (1점)

@pytest.mark.django_db
def test_get_book_by_id_not_found():
    # When & Then
    with pytest.raises(##) as exc_info: # 여기에 동작 코드를 작성하세요 (1점)
        #(9999) # 여기에 동작 코드를 작성하세요 (1점)

    assert "ID 9999에 해당하는 책이 없습니다." in str(exc_info.value)
```

✓ 설명

테스트 함수	설명
<code>test_get_book_by_id_success</code>	실제 Book 객체를 만든 후, 해당 ID로 조회가 되는지 확인
<code>test_get_book_by_id_not_found</code>	존재하지 않는 ID를 조회할 때 <code>BookNotFound</code> 예외가 발생하는지 검증

`@pytest.mark.django_db`는 DB 접근이 필요한 테스트에 필수입니다.

✓ 실행 명령어

```
pytest library/tests/
```

or 전체 테스트

```
pytest
```

✓ 요약

항목	설명
테스트 위치	<code>library/tests/test_services.py</code>
테스트 대상	서비스 함수 (<code>get_book_by_id</code> , <code>get_borrow_history...</code>)
장점	뷰/템플릿 독립, 빠르고 안정적인 단위 테스트 가능
프레임워크	<code>pytest</code> , <code>pytest-django</code> 권장

`get_borrow_history_for_book(book)` 함수에 대한 단위 테스트.

이 함수는 책(Book)에 연결된 대출 이력(BorrowHistory)을 조회하며, 이력이 없을 경우 예외를 발생시키는 구조입니다.

✓ 1. 서비스 함수 (리마인드)


```
# services/book_service.py

from library.exceptions import BookHasNoBorrowHistory

def get_borrow_history_for_book(book):
    history_qs = book.borrow_history.order_by('-borrowed_at')
    if not history_qs.exists():
        raise BookHasNoBorrowHistory(f"'{book.title}' 도서에는 대출 기록이 없습니다.")
    return history_qs
```

✓ 2. 테스트 대상 행동

조건	기대 결과
대출 기록이 존재	QuerySet 반환
대출 기록이 없음	<code>BookHasNoBorrowHistory</code> 예외 발생

✓ 3. 테스트 코드 (`tests/test_services.py`) : 총 9점

```
import pytest
from django.contrib.auth.models import User
from library.models import Book, BorrowHistory
from library.services.book_service import get_borrow_history_for_book
from library.exceptions import BookHasNoBorrowHistory

@pytest.mark.django_db
def test_get_borrow_history_for_book_success():
    # Given
    user = User.objects.#(username='testuser') # 여기에 동작
    코드를 작성하세요 (1점)
    book = Book.objects.#(title='Test Book', author='Tester', i
    sbn='1234567890123') # 여기에 동작 코드를 작성하세요 (1점)
    BorrowHistory.objects.#(book=book, user=user) # 여기에
    동작 코드를 작성하세요 (1점)

    # When
```

```

histories = #_____#(book) # 여기에 동작 코드를 작성하세요 (1점)

# Then
assert histories.#_____# == 1 # 여기에 동작 코드를 작성하세요 (1점)
assert histories.first().#_____# == user # 여기에 동작 코드를 작성
하세요 (1점)

@pytest.mark.django_db
def test_get_borrow_history_for_book_no_history():
    # Given
    book = Book.objects.#_____#(title='Empty Book', author='Nobod
y', isbn='9999999999999') # 여기에 동작 코드를 작성하세요 (1점)

    # When & Then
    with pytest.raises(#_____#) as exc_info: # 여기에 동작 코드를 작성
하세요 (1점)
        #_____#(book) # 여기에 동작 코드를 작성하세요 (1점)

    assert '도서에는 대출 기록이 없습니다.' in str(exc_info.value)

```

✓ 4. 설명

- `BorrowHistory` 가 존재하면: 정렬된 QuerySet을 반환하고 `.count()`, `.first()` 로 검증
- 존재하지 않으면: `BookHasNoBorrowHistory` 예외를 명확히 캐치

✓ 5. 실행 명령어

pytest library/tests/test_services.py

✓ 정리

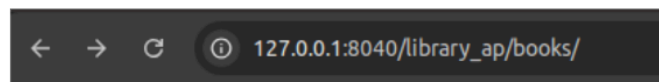
테스트 항목	기능
성공 케이스	1개 이상의 대출 이력 반환
실패 케이스	대출 이력 없음 → 예외 발생

13. 과제 결과 화면

✓ 결과 확인

URL	기능
<code>/library/books/</code>	전체 책 목록 보기
<code>/library/books/1/history/</code>	특정 책의 대출 이력 확인

- 전체 책 목록 보기 화면 캡처 : 1점



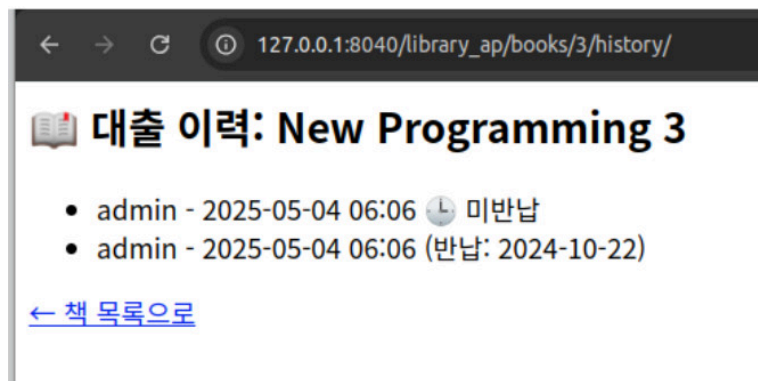
책 목록

- [Room Programming 1](#)
- [He Programming 2](#)
- [New Programming 3](#)
- [Speech Programming 4](#)
- [Challenge Programming 5](#)
- [Task Programming 6](#)
- [Wife Programming 7](#)
- [Space Programming 8](#)
- [Career Programming 9](#)
- [Product Programming 10](#)
- [Little Programming 11](#)
- [Million Programming 12](#)
- [Tonight Programming 13](#)
- [Subject Programming 14](#)
- [Eight Programming 15](#)
- [Bring Programming 16](#)
- [Will Programming 17](#)
- [Get Programming 18](#)
- [Husband Programming 19](#)
- [Fire Programming 20](#)
- [Charge Programming 21](#)
- [Down Programming 22](#)
- [Type Programming 23](#)

- 특정 책의 대출 이력 확인 (대출 이력이 없는 경우 화면 캡처) : 1점



- 특정 책의 대출 이력 확인 (대출 이력이 있는 경우 화면 캡처) : 1점



- 단위 테스트 결과 성공 화면 캡처 : 1점

```
(base) sckubunu20@sckubunu20-ThinkPad-X1-Carbon-Gen-8:~/webserver/class_prep/library$ pytest library_ap/tests/
===== test session starts =====
platform linux -- Python 3.12.7, pytest-7.4.4, pluggy-1.0.0
django: version: 5.1.7, settings: library.settings (from ini)
rootdir: /home/sckubunu20/webserver/class_prep/library
configfile: pytest.ini
plugins: django-4.11.1, Faker-37.0.2, anyio-4.2.0, langsmith-0.3.33
collected 4 items

library_ap/tests/test_services.py .... [100%]

===== 4 passed in 0.27s =====
```

- 제출
 - 과제 9 총점 : 총 40점

- 깃허브 제출 링크 : 1점
- 코드 완성 : 39점
 1. 뷰(View) 와 서비스(Service Layer) 분리 이점 : 총 6점
 - ✓ 2. 서비스 코드 (services/book_service.py) : 3점
 - ✓ 3. 뷰 코드 (views.py) — 서비스 호출만 수행 : 3점
 2. 뷰(View) 와 서비스(Service Layer) 분리 예제 : 총 8점
 - ✓ URL 설정 — library/urls.py : 3점
 - ✓ 뷰 구현 — views.py : 3점
 - ✓ 템플릿 생성 : 2점
 3. 예외 처리 포함 서비스 함수 : 총 6점
 - ✓ 서비스 레이어 예제 (services/book_service.py) : 4점
 - ✓ 뷰 예제 (views.py)에서의 예외 처리 : 2점
 4. 단위 테스트 : 총 15점
 - ✓ 테스트 코드 예시: tests/test_services.py : 6점
 - ✓ 3. 테스트 코드 (tests/test_services.py) : 9점
 5. 결과 확인 : 총 4점
 - ✓ 전체 책 목록 보기 화면 캡처 : 1점
 - ✓ 특정 책의 대출 이력 확인 (대출 이력이 없는 경우 화면 캡처) : 1점
 - ✓ 특정 책의 대출 이력 확인 (대출 이력이 있는 경우 화면 캡처) : 1점
 - ✓ 단위 테스트 결과 성공 화면 캡처 : 1점
- 5월 28일 (수) 13:30분까지
- 늦게 제출은 인정하지 않습니다.
 - 코드 안의 # _____ # 밑줄 앞 뒤 # # 는 제거하고 제출바랍니다.
- 제출 방식
 - 노션이나 hwp, 워드 등을 사용해서 제출 바랍니다.
 - 가상강의실 제출