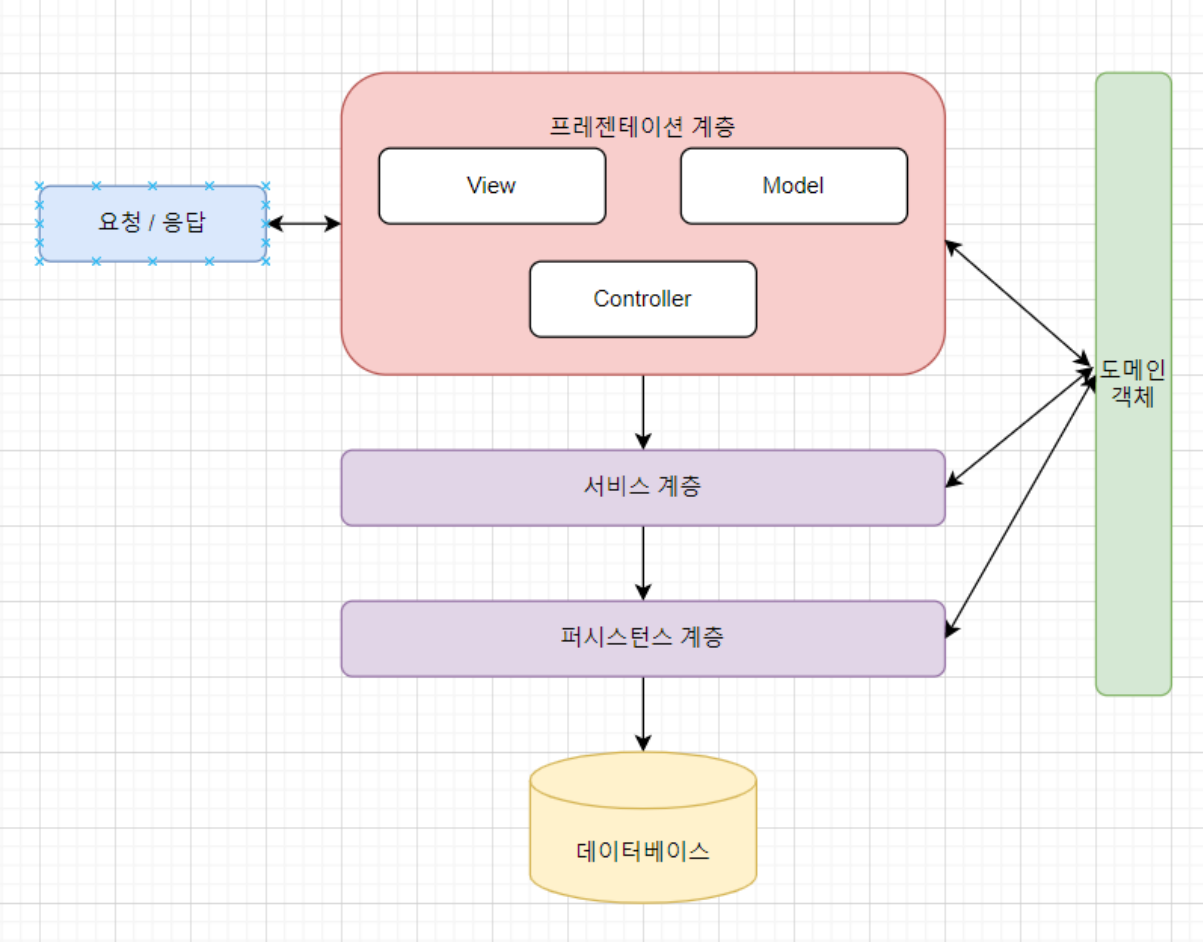
Class 230227

Spring - 1일차

계층적 구조

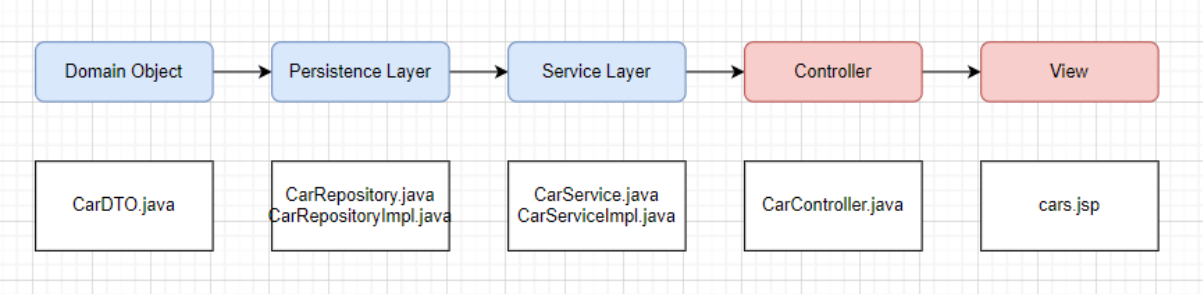


도메인 객체 (domain object)	데이터 모델 객체 정보를 저장. jsp 에서의 DTO와 같
퍼시스턴스 계층 (persistence layer)	데이터 엑세스 계층 데이터베이스에 접근하여 데이터를 처리. jsp DAO와 같다.
서비스 계층 (service layer)	비지니스 계층 애플리케이션에서 제공하는 포괄적인 서비스를 표현. 프레젠테이션 계층과 퍼시스턴스 계층을 연결하는 역할.
프레젠테이션 계층	사용자에게 데이터를 입력 받거나 결과를 서버에 전달하고 사용자에게

(presentation layer)	보여주는 계층.

- 계층적 구조의 작업이 필요한 이유
  - 코드의 복잡성 증가
  - 유지 보수의 어려움
  - 유연성 부족
  - 중복 코드 증가
  - 낮은 확장성

CarShop 계층 구조



▼ CarDTO.java

```
package com.carshop.domain;

import lombok.Data;

@Data
public class CarDTO {
  private String cid;
  private String cname;
  private String cprice;
  private String ccate;
  private String cdesc;
}
```

▼ CarRepository.java

```
package com.carshop.repo;

import java.util.List;

import com.carshop.domain.CarDTO;

public interface CarRepository {
  public List<CarDTO> getAllCarList();
}
```

▼ **CarRepositoryImpl.java**

```
package com.carshop.repo;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;

import com.carshop.domain.CarDTO;

@Repository
public class CarRepositoryImpl implements CarRepository {
  private List<CarDTO> listOfCars = new ArrayList<CarDTO>();

  public CarRepositoryImpl() {
    CarDTO car1 = new CarDTO("c0001", "아반테", "2500", "승용차", "거의새거");
    CarDTO car2 = new CarDTO("c0002", "소나타", "3000", "승용차", "새거");
    CarDTO car3 = new CarDTO("c0003", "그랜저", "3500", "승용차", "중고");

    listOfCars.add(car1);
    listOfCars.add(car2);
    listOfCars.add(car3);
  }

  @Override
  public List<CarDTO> getAllCarList() {
    // TODO Auto-generated method stub
    return listOfCars;
  }
}
```

Python Advence - Crawling

▼ 음악 차트 순위 크롤링

```
# 벅스 뮤직 차트 스크래핑

import requests                      # html 문서 텍스트 형태로 가져옴
from bs4 import BeautifulSoup as bs # html 문서 태그 접근 가능하게 변환
import pandas as pd
from selenium import webdriver      # 크롤링 불가능한 서비스 가능하게 함

'''
벅스 차트
'''
html = requests.get('https://music.bugs.co.kr/chart')
bsObj = bs(html.text, "html.parser")

titles = bsObj.select('.byChart > tbody > tr th p.title > a')
singers = bsObj.select('.byChart > tbody > tr td p.artist > a')
ranks = bsObj.select('.byChart > tbody > tr td > div > strong')

songData = []
for i in range(0, 100):
    title = titles[i].text
    singer = singers[i].text
    rank = ranks[i].text
    songData.append([rank, title, singer])
df = pd.DataFrame(songData, columns=['순위', '타이틀', '가수'])
print(df)


'''
멜론 차트
'''
driver = webdriver.Chrome('chromedriver.exe')           # 크롬을 이용해 불러옴
driver.get('https://www.melon.com/chart/index.htm')

txt = driver.page_source
html = bs(txt)
titles1 = html.select('tbody .rank01 a')
singers1 = html.select('tbody .rank02 a')
ranks1 = html.select('tbody .rank')

songData1 = []
for i in range(0, 100):
    title = titles1[i].text
    singer = singers1[i].text
    rank = ranks1[i].text
    songData1.append([rank, title, singer])
df1 = pd.DataFrame(songData1, columns=['순위', '타이틀', '가수'])
print(df1)


'''
지니 차트
'''
driver = webdriver.Chrome('chromedriver.exe')           # 크롬을 이용해 불러옴
driver.get('https://www.genie.co.kr/chart/top200')

txt = driver.page_source
```

```
html = bs(txt)
titles2 = html.select('tbody .info .title')
singers2 = html.select('tbody .info .artist')
ranks2 = html.select('tbody .number')

songData2 = []
for i in range(0, 50):
    title = titles2[i].text.strip()
    singer = singers2[i].text.strip()
    rank = ranks2[i].text[0:3].strip()
    songData2.append([rank, title, singer])
df2 = pd.DataFrame(songData2, columns=['순위', '타이틀', '가수'])
print(df2)
```

`select()` 와 `find()`

공통 기능

- 두가지 모두 태그를 찾아주는 메서드

사용

- 괄호 안의 조건에 해당하는 태그를 모두 추출
  - `.select()`
  - `.find()_all`
- 괄호 안의 조건에 해당하는 태그를 하나만 추출
  - `.select_one()`
  - `.find()`

차이

- `.find()`는 참/거짓 조건을 넣어서 필터링 가능