캐싱의 목적은 다음과 같습니다.
> DB 부하 감소
> 리소스 절약
> 성능 향상

그러면 어떤 데이터들을 캐싱처리하는게 좋을까? 를 생각한다면
> 자주 변경되지 않는 데이터
> 자주 조회되는 데이터
> 제공하기 위한 계산이 복잡한 데이터 (통계 등..)
라고 생각해 볼 수 있을 것 같습니다.

그럼 이걸 콘서트 프로젝트에 적용해본다면.. ? (조회가 자주 되는 데이터 기능들..)
> 콘서트 목록 조회
: 콘서트 목록은 자주 바뀌지 않는 데이터가 맞고,
자주 바뀌어봐야 시간 단위 아니면 데일리 단위로 바뀌는 데이터 일 것 같습니다.
추가로 조회 또한 가장 자주 일어나는 데이터 중 하나이므로 캐싱처리하기 적합합니다.

> 콘서트 날짜 조회
: 콘서트 목록과 같은 맥락으로 자주 바뀌지 않는 데이터이며, 조회또한 자주 일어납니다. 따라서 해당 데이터도 캐싱처리하기 적합합니다.

> 좌석 조회
: 좌석 조회도 자주 조회되는 데이터입니다. 하지만 인기 있는 콘서트의 경우 초반에 사람이 한번에 몰리기도 하고 그만큼 데이터 변경이 매우 빠르게, 자주 일어나는 데이터이기 때문에 캐싱처리에는 부적합하다고 생각합니다.

> 콘서트 목록 조회

테스트 코드는 아래와 같습니다.

(service 에 ForTest() 메서드를 캐시처리 없이 하나 더 만들었습니다.)

```
@Test
void 콘서트조회_캐싱처리_속도비교() throws Exception {
    //캐싱을 위해 호출
    concertService.getConcerts();
    Thread.sleep( millis: 1000);

    //캐싱된걸 호출
    log.info("캐싱된 목록 조회 시작!");
    Long startTime = System.currentTimeMillis();
    concertService.getConcerts();
    Long endTime = System.currentTimeMillis();
    log.info("캐싱된 목록 조회 끝!");
    log.info("캐싱된 목록 조회 소요 시간: {}", (endTime - startTime) + "ms");

    log.info("캐싱없는 목록 조회 시작!");
    startTime = System.currentTimeMillis();
    concertService.getConcertsNoCacheForTest();
    endTime = System.currentTimeMillis();
    log.info("캐싱없는 목록 조회 끝!");
    log.info("캐싱없는 목록 조회 소요 시간: {}", (endTime - startTime) + "ms");
}
```

> 결과

```
main] c.h.c.i.ConcertIntegrationTest           : 캐싱된 목록 조회 시작!
main] c.h.c.i.ConcertIntegrationTest           : 캐싱된 목록 조회 끝!
main] c.h.c.i.ConcertIntegrationTest           : 캐싱된 목록 조회 소요 시간: 4ms
main] c.h.c.i.ConcertIntegrationTest           : 캐싱없는 목록 조회 시작!
main] c.h.c.i.ConcertIntegrationTest           : 캐싱없는 목록 조회 끝!
main] c.h.c.i.ConcertIntegrationTest           : 캐싱없는 목록 조회 소요 시간: 25ms
```

> 콘서트 날짜 조회

테스트 코드는 아래와 같습니다.

(service 에 ForTest() 메서드를 캐시처리 없이 하나 더 만들었습니다.)

```
@Test
void 날짜조회_캐싱처리_속도비교() throws Exception {

    //캐싱을 위해 호출
    concertService.getDates( concertId: 1L);
    Thread.sleep( millis: 1000);
    //캐싱된걸 호출
    log.info("캐싱된 목록 조회 시작!");
    Long startTime = System.currentTimeMillis();
    concertService.getDates( concertId: 1L);
    Long endTime = System.currentTimeMillis();
    log.info("캐싱된 목록 조회 끝!");
    log.info("캐싱된 목록 조회 소요 시간: {}", (endTime - startTime) + "ms");

    log.info("캐싱없는 목록 조회 시작!");
    startTime = System.currentTimeMillis();
    concertService.getDatesNoCacheForTest( concertId: 1L);
    endTime = System.currentTimeMillis();
    log.info("캐싱없는 목록 조회 끝!");
    log.info("캐싱없는 목록 조회 소요 시간: {}", (endTime - startTime) + "ms");
}
```

> 결과

```
c.i.ConcertIntegrationTest           : 캐싱된 목록 조회 시작!
c.i.ConcertIntegrationTest           : 캐싱된 목록 조회 끝!
c.i.ConcertIntegrationTest           : 캐싱된 목록 조회 소요 시간: 4ms
c.i.ConcertIntegrationTest           : 캐싱없는 목록 조회 시작!
c.i.ConcertIntegrationTest           : 캐싱없는 목록 조회 끝!
c.i.ConcertIntegrationTest           : 캐싱없는 목록 조회 소요 시간: 17ms
```

두 기능 모두 테스트를 통해 데이터가 많아봐야 10~100건인데도 불구하고
속도차의가 유의미하게 나는 것을 확인할 수 있었습니다.