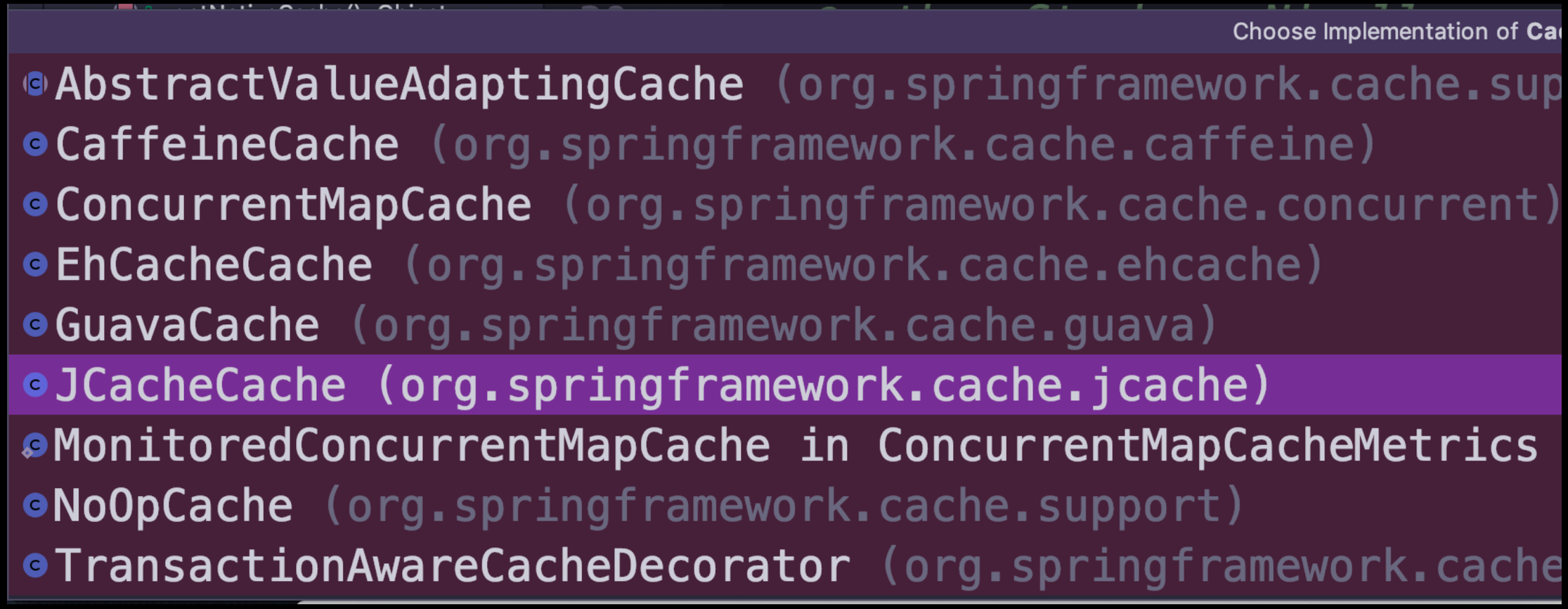
Spring cache

Caching

- Spring framework는 caching기능을 지원.
    - Cacheing 방법에 대한 추상화를 지원, 설정을 편하게 함.
    - 스프링 부트 또한 cache에 대한 auto-configure를 지원 (@EnableCaching 어노테이션을 통해)
    - Cache에 대한 third-party dependency를 추가하지 않았다면, Springboot는 Concurrent Map을 지원한다.

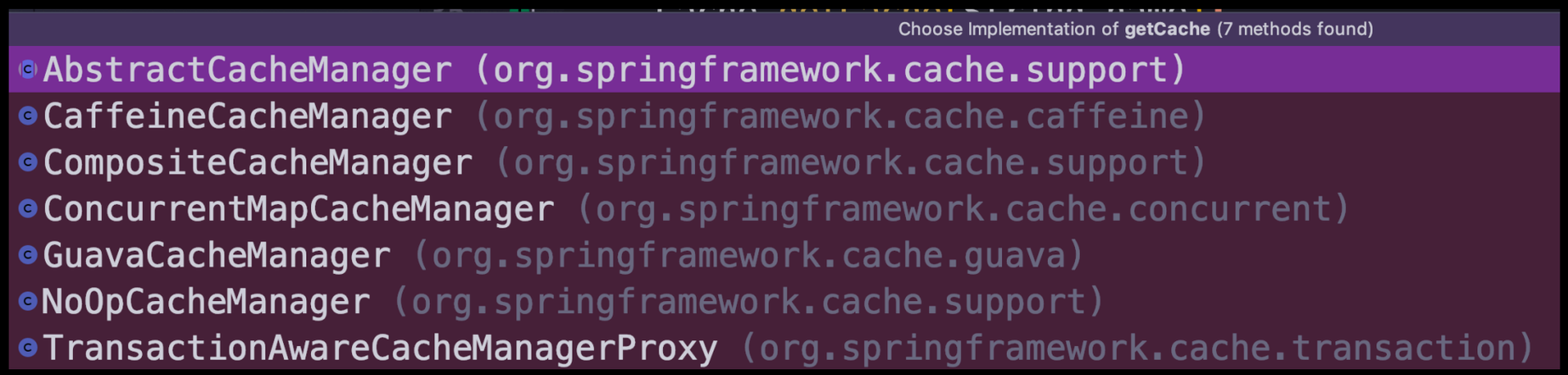
Spring Cache

- Common Cache operation에 대한 interface
- 여러 구현 Cache들을 지원



CacheManager

- Spring의 Cache관련 Service Provider Interface
- @EnableCaching Annotation을 사용하기 위해서는 CacheManager가 빈으로 등록되어야 한다.



LoadingCache

- Cache Miss가 발생하면 자동으로 데이터 로드(기본 Cache는 데이터를 자동으로 로딩하지 않는다.)

- LoadingCache에서 get을 할때, 해당 key에 해당하는 Value가 없으면 load, loadAll Method를 호출하여, key에 해당하는 데이터를 채움.

CacheLoader

- LoadingCache를 만들기 위해 사용하는 객체로서, Key를 기반으로 Value를 갖고오는 기능을 수행

- LoadingCache에서 Key가 발견되지 않으면 CacheLoader객체를 통해 Cache를 로딩한다.

- LoadingCache를 만들때, CacheLoader를 필수적으로 파라미터로 넘겨줘야 된다.

FlexibleCaffeineCacheManager

- CaffeineCacheManager Class를 상속 받고, InitializingBean을 구현한 객체 (InitializingBean을 구현한 이유는 모든 객체들이 생성되고 따로 세팅을 하기 위하여 구현한 것으로 이해됨)

- Spring에서 제공하는 CaffeineCacheManager는 한개의 CaffeineCache만 정의할 수 있는 제한이 있다는 단점이 존재. —> Cache한개마다 스팩을 정의해주는 CacheManager를 만들어야 됨.

- FlexibleCaffeineCacheManager는 다이나믹하게 미리 정의된 캐시에 대한 스팩을 정의할 수 있고, 단일 스팩으로도 정의가 가능하다.

```java
@Override
public void afterPropertiesSet() {
    for (var cacheSpecEntry : cacheSpecs.entrySet()) {
        //noinspection ObjectAllocationInLoop
        builders.put(cacheSpecEntry.getKey(), Caffeine.from(cacheSpecEntry.getValue()));
    }
}
```

```yaml
# example
#caffeine:
#  specs:
#    a-cache: maximumSize=5000,expireAfterWrite=200s
#    b-cache: maximumSize=1000,expireAfterWrite=100s
```