**EH CACHE 3**

-Introduction

* Open Source library implemented in java for implementing caching in java programs,
* Involves storing of data in cache for faster retrieval,
* Caching is useful in scenarios where same data is requested again and again,
* In spring boot entire caching infrastructure is configured for us which is activated by using annotation @EnableCaching
* Spring caching service is abstraction not implementation.
* Spring provides many cache providers
  + EhCache 3
  + Redis
  + Hazelcast
  + Couchbase

Spring itself uses default cache provider which caches in main memory using maps

-Eh Cache tiers

* It stores data in tiers with top tier is called as near tier and lowest level tier is called as authority tier,
* Most frequently used data is stored in near tier and authority tier contains all cached data,
* Three memory areas are supported by ehCache
  + On-heap Store – uses java heap memory to store cache, this memory is scanned by garbage collector and is very fast but limited
  + Off-heap store – Uses ram to store cache entries, not subjected to garbage collection and is quite fast but slower than on-heap store. Cache entries are moved from here to heap memory which makes it slow.
  + Disk store- Stores data on disk and is much slowest of all.

-Understanding ehcache.xml

* In the persistence tag, we define the directory for a file-based cache on the hard disk (disk store). This is only the definition of the folder. Whether we really want to use a disk store or not will be configured later.
* In the expiry tag, we define a *time to live* (ttl) of 30 seconds. The time to live specifies how long a cache entry may remain in the cache independently of access. It is also possible to define a *time to idle* (tti). The time to idle specifies how long the cache entry may exist in the cache without access.
* In the listeners tag, we configure a CacheEventListener.
* In resource tag we configure tiers and specify capacity,
* In cache tag we specify our cache, it’s key and value-type(return type)
* In case of method accepting multiple parameters key-type will be java.util.arraylist
* In case if you don’t provide key it takes all method params as key

@CachePut

For updating cache value

@CacheEvict

* For clearing cache
* We can either clear whole cache or we can clear one element from cache (in delete event)
* We can also run scheduler for method in using CacheManager for clearing out all caches after specified time inside CacheConfig class annotated with @Configuration annotation

@Scheduled(fixedRate = 6000)

public void evictAllcachesAtIntervals() {

evictAllCaches();

}

public void evictAllCaches() {

cacheManager.getCacheNames().stream()

.forEach(cacheName -> cacheManager.getCache(cacheName).clear());

}