(Design Multithreaded Cache)

Eviction Policies.

(LRU
LFU)

(Having Unique Key-Value Pairs)

→ put() → store data
→ get() → retrieve data
→ delete() → when capacity full eviction.

**(Concurrent access should be done)

*) Take care of Background thread handling the process of eviction, if size of queue is getting full then evict it.

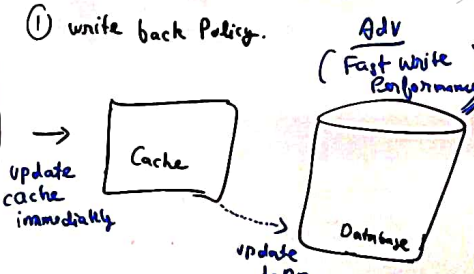
*) Optional time based expiration, again a background thread running the TTL process.

After Discussion with Candidate & Interviewer.

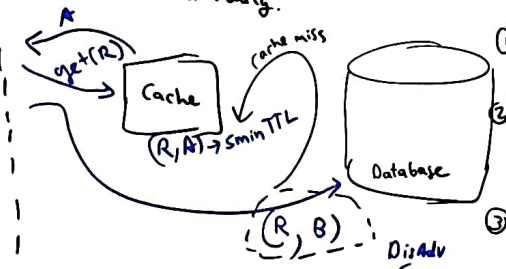
* System will store & retrieve K-V pairs
* Limited Memory → Proper Eviction Policy. (only LRU don't do LFU)
* 2 Core Operations
  → Read
  → write.
* Java Generics will be used
* thread Safe and handle concurrent operations

Handling concurrent operations on cache
(Basically different Write Policies.)

① write back Policy.



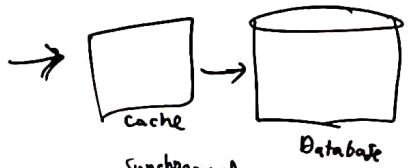
② Write Around Policy.



① Write directly goes to the DB.
② When → get(R) Call comes to cache, then at that point A is returned.
③ When entry expires after 5 mins, then (R, B) is loaded after cache miss.

DisAdv (Stale Data till expiration)

③ Write through Cache



DisAdv (Adv)
Slow (Consistent Persistance)

Synchronously write to DB and don't send Ack before it's completes.

(Design Multithreaded Cache)

(New Concept Learnt)

(Out of Scope. Just Like Hello Interview.)

→ Request Collapsing

→ Hot Loading

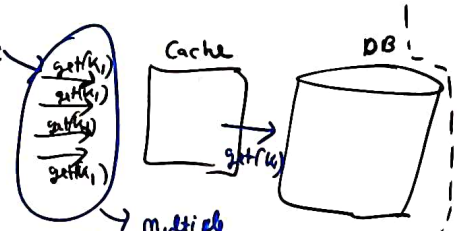
Load hot keys, during server startup itself.

→ Event Logging

↳ Not Logging cache hit/miss.

→ Read Your Own writes →

Well Basically we will not be handling this separately as a consistency issue and let ConcurrentHashMap take care of it



→ Multiple requests are coming for the same key then collapse the request & dont bombard the DB & and send only one key.

(Classes & Interfaces)

Storage Interfaces

CacheStorage<K,V>

- put()
- get()
- remove()
- containsKey()
- size()
- getCapacity()

DBStorage<K,V> (persistent storage).

- write
- read
- delete

InMemoryCacheStorage<K,V> implements CacheStorage<K,V> {

Map<K,V> cache = new ConcurrentHashMap<>()

→ put() → containsKey() → getCapacity()
→ get() → size()

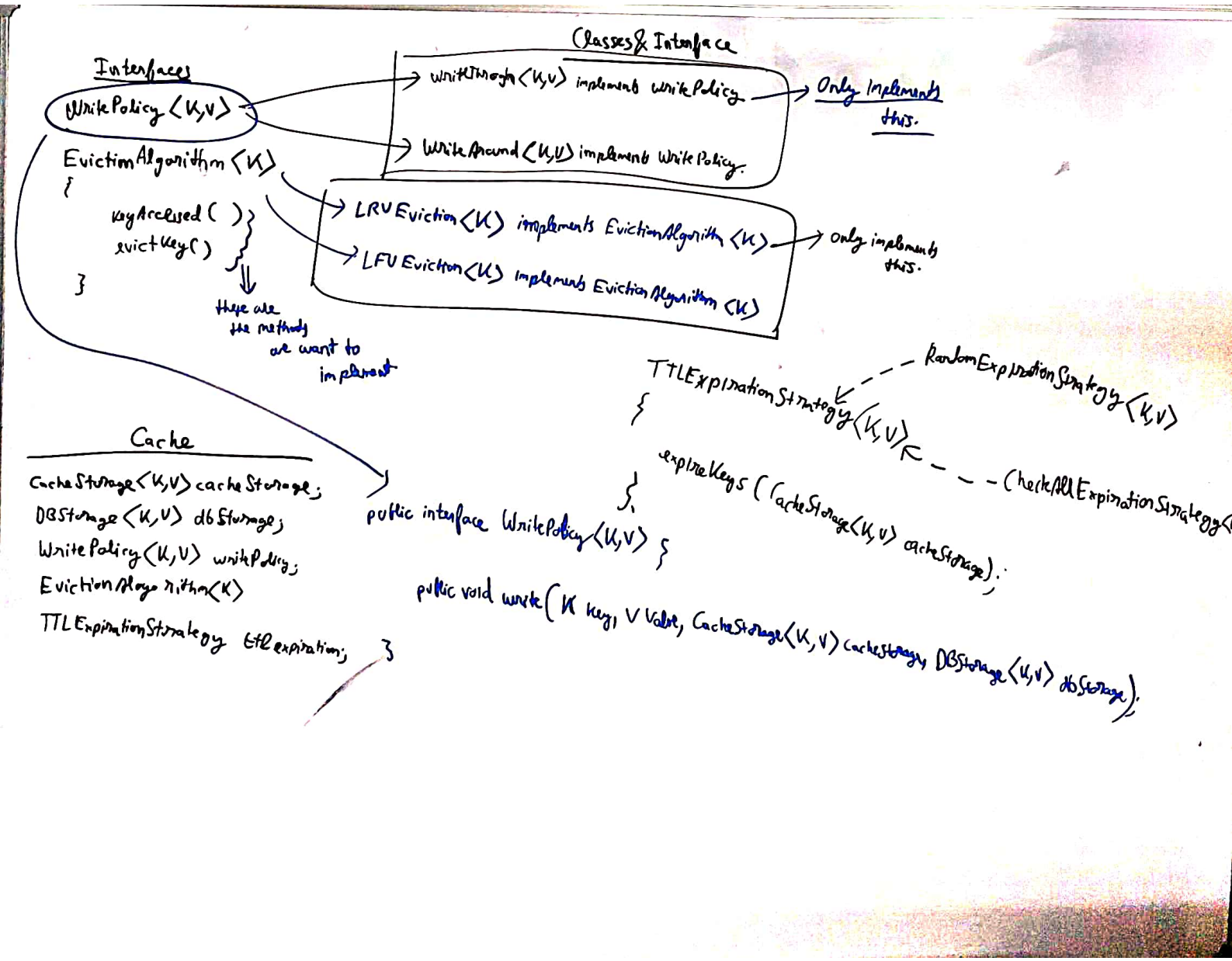
}

SimpleDBStorage<K,V> implements DBStorage<K,V> {

Map<K,V> database = new ConcurrentHashMap<>();

→ write() → read() → delete()

}


Interfaces
WritePolicy<K,V>
EvictionAlgorithm<K>
{
keyAccessed()
evictKey()
}
these are the methods we want to implement
Classes & Interface
WriteThrough<K,V> implements WritePolicy
WriteAround<K,V> implements WritePolicy.
Only implements this.
LRUEviction<K> implements EvictionAlgorithm<K>
LFUEviction<K> implements EvictionAlgorithm<K>
only implements this.
Cache
CacheStorage<K,V> cacheStorage;
DBStorage<K,V> dbStorage;
WritePolicy<K,V> writePolicy;
EvictionAlgorithm<K>
TTLExpirationStrategy ttlExpiration;
public interface WritePolicy<K,V> {
public void write(K key, V value, CacheStorage<K,V> cacheStorage, DBStorage<K,V> dbStorage);
}
TTLExpirationStrategy<K,V>
{
expireKeys(CacheStorage<K,V> cacheStorage);
}
RandomExpirationStrategy<K,V>
CheckAllExpirationStrategy<

Classes & Interface

```
class WriteThroughPolicy<K,V> implements WritePolicy<K,V> {

    public void write(K key, V value, CacheStorage<K,V> cacheStorage, DBStorage<K,V> dbStorage) throws InterruptedException {
        // do both the write Asynchronously
        CompletableFuture<Void> cacheFuture = CompletableFuture.runAsync(() -> {
            try {
                cacheStorage.put(K,V);
            }
            catch(Exception e){
                Thread.currentThread().interrupt();
            }
        });

        CompletableFuture<Void> dbFuture = CompletableFuture.runAsync(() -> {...write to DB});

        CompletableFuture.allOf(cacheFuture, dbFuture).join();
    }
```

Classes & Interface

Cache

CacheStorage<K,V> cacheStorage; → new InMemoryCacheStorage<K,V>

DBStorage<K,V> dbStorage; → new SimpleDBStorage<K,V>

WritePolicy<K,V> writePolicy; → new WriteThroughPolicy<K>

EvictionAlgo<K> evictionAlgo; → new LRUEvictionAlgo<K>

TTLExpiration<K> ttlExpiration; → new SimpleTTLExpirationPolicy

Writing Test Cases for Eviction, Normal writing, Persisting DB writing