Getting started

Pre Requirements

In order to use N Way Set Associative Cache you will need the following to be installed on your machine:

1. Java 8 (JDK 8) http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html
2. For a complete Java installation manual, please refer to: https://www.java.com/en/download/help/download_options.xml
3. Maven , for installation please refer to https://maven.apache.org/download.cgi

Generate the jar file

1. Use your favorite console client and navigate to the root folder
2. Run 'mvn package'

Now, after all completed successfully, the jar will be located under the 'target' folder – 'nway-1.0-SNAPSHOT.jar'

In order to use the library, you may need to add the jar to your clas.spath

Usage

Building new N Way Set Associative cache

For creating a new default n way instance simply type:

NWayAssociativeCache<String, Integer> basicNWayCache = new NWayAssociativeCache.Builder<String, Integer>(NUM_SETS, NUM_LINES).build();

Where:

- <String, Integer> can be any <Key, Value> combination
- NUM_SETS : can be any number (>0) that indicates number of sets
- NUM_LINES: can be any number (>0) that indicates number of lines inside each sets

Change Eviction Policy

By default, N Way Cache comes with a default LRU eviction policy.

Change to MRU eviction policy

MRU algorithm is available as well, In order to use MRU use the 'withEvictionPolicy' method and provide a new MRUEvictionPolicy instance

NWayAssociativeCache<String, Integer> cacheWithNewEviction = new NWayAssociativeCache.Builder<String, Integer>(NUM_SETS, NUM_LINES)

.withEvictionPolicy(new MRUEvictionPolicy<>()).build();

Change to any eviction policy

You can use any eviction policy implementation that you desire by only providing your own implementation to the *EvictionPolicy* interface (@FunctionalInterface)

```
@FunctionalInterface
public interface EvictionPolicy <K,V>{

    /**
     * the eviction method
     * @param entries the set entries.
     */
    void evict(List<CacheEntry<K, V>> entries);

}
```

You can use a lambda expression inside the 'withEvictionPolicy' method

Example:

```
NWayAssociativeCache<String, Integer> cacheWithNewEviction = new
NWayAssociativeCache.Builder<String, Integer>(2,2)
        .withEvictionPolicy(list->{
            //----
            //Your implementation here
            //----
        })
        .build();
```

Use Cache Outlet

By default, N-way Cache doesn't come with an implementation for communicating with any backing store for retrieving values. Meaning, when a client tries to fetch a value corresponding to a given key and the key is missing from the cache – the cache will return NULL w/o trying to look for the value in a backing store.

In case you want to change the default behavior, you simply need to provide your own implementation for the *CacheOutlet* functional interface (@FunctionalInterface) using the method withCacheOutlet(/*your impl*/) within the cache build process.

```
@FunctionalInterface
public interface CacheOutlet<K,V> {

    /**
     * this methos is expected to direct the call to a third party data source for
obtaining a value for the given key
     * @param key the key
     * @return
     */
    V obtain(K key);

}
```

Example:

```
NWayAssociativeCache<String, Integer> cacheWithNewEviction = new
NWayAssociativeCache.Builder<String, Integer>(2,2)
        .withCacheOutlet(key->{

            //----
            //Your implementation he
            //----

            return myValue;
        })
        .build();
```

Use the Cache

After initiating the cache you can perform 3 main actions: PUT\GET\REMOVE

All actions are contained inside the Cache<K,V> interface:

```
public interface Cache <K, V> {
    /**
     * get the value
     * @param key the key
     * @return the value
     */
    V get(K key);

    /**
     * put a new entry (key->value) in cache
     * @param key the key
     * @param value the value
     */
    void put(K key, V value);

    /**
     * removes the entry corresponding to the given key
     * @param key the key
     */
    void remove(K key);
}
```

Examples:

```
NWayAssociativeCache<String, Integer> cache = new NWayAssociativeCache.Builder<String,
Integer>(1,2).build();


cache.put("1", 4);

cache.get("1")

cache.remove("1");
```

Design

The N Way Set Associate cache, build in such a way that it contains predefined number of sets where each set contains 'N' cache entries.

Cache

Contains an immutable list of cache sets to support multi-threading environment

Cache Set

Each set contains the followings:

- Double linked list of cache entries – immutable and constructed only on the main cache initialization.
- Read/write lock – allows us to manage all major concurrency within the set, we will use wrote lock for the followings:
    - When creating and adding a new entry
    - Within eviction process, when entries are removed

  The read lock is used for all the other actions as:

    - Trying to obtain an entry

  Important to recall that read locks will not hold other read actions. Only write locks should lock any other read/write actions.

- Age Bit generator – an atomic field that guarantee a proper increment in a concurrent environment

Cache Entry

Each entry contains the followings:

- The Key
- The Value
- The last touched time - is updated while creating\reading an entry
- The creating Time – when this entry was created
- Age Bit – in similar to 'touched time' the age bit is updated in every creation/reading of an entry – it allows us to easily implement multiple Eviction policies as LRU or MRU