N-Way Set Associative Cache Design.

Interfaces Exposed

<IcachePolicy>

- ⇨ Each new cache Policy will need to implement this interface.

<IcachePolicyFactory>

- ⇨ A CachePolicyFactory will implement this interface, to create a multiple policy objectsf for each of the cacheSets.

<IcacheSetHashCalculator>

- ⇨ Every calculator to calculate the setIndex, looking at the key for the item will have to implement this interface

<class> AssociativeCache

- ⇨ Gets depependenies injected
    1. CachePolicyFactory – A factory to create cache policies for each of the cache sets. *If not supplied a default LRU CachePolicy will be created for each set*
    2. Way – an Integer to indicate the number of cache sets.
    3. HashSetCalculator – A HashCaclulator Implementation to calculate hashing on keys to get the appropriate cache set. *If not suppiled, a default HashCalculator that will look at inbuilt hascode of keys to generate the setIndex for the Set.*
    4. Takes in an instance of appender skeleton, to use it as a logger.
- ⇨ Composes in itself a list of CacheSets called CacheSetLookup.
- ⇨ Composes in itself an Appender for logging purposes
- ⇨ Composes a HashSetCalculator to calculate set indiexes, based on keys.
- ⇨ <Methods Supported>
    1. Add (Key, Value)
        - Gets the set index
        - Adds an item in the cache set

- Notifies the CachePolicy to update the position of the item

2. Remove (Key)
    - Get the set index
    - Removes the item with its key from the index
    - Deletes the item from the cache policy

3. Clear()
    - For each cache set in the cacheSetLookup
        - Clears all the items in the cache set.
4. Get (K Key)
    - Get the Set Index
    - Retrieves the value from the Key by doing a lookup in the set.

5. GetCacheSets()
    - Gets All the Cache Sets that were initialized in the cache.

<class> CacheSet

⇨ Gets depependencies injected such as the SetCapacity

⇨ Cache Policy for the particular CacheSet

⇨ Appeneder for Logging Purposes.

⇨ Composes in it itself a dictionary to look up CacheItems based on keys.

⇨ <Methods Supported>

** All the below methods are behind a lock to ensure that only one item is updated added at a time, by a thread **

1. `AddItemToCacheSet` (Key, Value)
    - Gets the Value from the CacheItem LookUp
    - Updates the look up for the item based on the policy it was supplied.

2. `RemoveItemFromCacheSet` (Key)
    - Removes the item with its key from the cacheItemLookup
    - Deletes the item from the cache policy

3. Clear()
    - For each cacheItem in the cacheSetLookup
        - Clears all the items from the policy

4. `AddItemToCacheSet` (K Key)
    - Adds the item in the cache set in the CacheItemLookUp
    - Updates the look up for the item based on the policy it was supplied.

5. `GetAllKeys` ()
    - Gets all the keys for the all items in the cache set.

<class> LRUCachePolicy

- ⇨ Implements a Queue to maintian the position of the nodes in the CacheSet.
- ⇨ Any new item added, or updated in moved to the front of the queue
- ⇨ If setCapacity is full, then on eviction, the last item is removed.

<class> MRUCachePolicy

- ⇨ Implements a Queue to maintian the position of the nodes in the CacheSet.
- ⇨ Any new item added, or updated in moved to the front of the queue
- ⇨ If setCapacity is full, then on eviction, the first item is removed.