Account cache

Your task is to provide an implementation for the `AccountCache` interface shown below. You need to implement all the methods of this interface according to documentation for these methods.

This is an in-memory cache for instances of the `Account` class also presented below.

You're not expected to use any database, ready-made caches or any other 3rd-party libraries - let's keep all the data in memory and use just the standard Java.

Your implementation of the `AccountCache` interface should work correctly in a multi-threaded environment with multiple threads potentially accessing any methods simultaneously.

The implementation should have a configurable limit on the maximum number of accounts to store in memory. If a new account is registered and there's already this maximum number of accounts in the cache, the cache should evict (remove) the least recently used account (the one that hasn't been accessed via the `getAccountById` or updated via `putAccount` for the longest time). Consider the implementation to be a simple example of an LRU cache. Example of the eviction algorithm in action:

1. Cache configured to contain at most 2 items
2. `putAccount(A)`
3. `putAccount(B)`
4. `putAccount(C)` → account A is automatically evicted
5. `getAccountById(B.id)` → 'use' the account B
6. `putAccount(D)` → account C is evicted

You're free to fix any issues / problems in the `Account` and `AccountCache` classes - if you see anything that can be improved in how these classes are defined, it's ok to fix it. This includes renaming of any methods and fields, amending Javadocs etc.

Consider overall efficiency of the implementation - we expect the cache to store many accounts and methods called frequently.

Please provide unit tests for your implementation.

```
public class Account {
    public long id;
    public long balance;
}

/**
```

```
* Keeps track of accounts in memory, providing methods to query the
* accounts by different criteria.
*/
public interface AccountCache {

    /**
     * @return account by its unique id
     */
    Account getAccountById(long id);

    /**
    * Registers a listener that will be notified when an account is
    * registered or updated via the {@link #putAccount(Account)}
    * method
    */
    void subscribeForAccountUpdates(Consumer<Account> listener);

    /**
     * @return top 3 accounts by balance, sorted from the largest balance
     * to the smallest one
     */
    List<Account> getTop3AccountsByBalance();

    /**
    * @return the number of 'hits' (when an account was found) to
    * the 'getAccountById' method of this service
    */
    int getAccountByIdHitCount();

    /**
     * Puts or updates an account in the service. If the cache is at
     * the full capacity, removes the least recently used account
     * (LRU policy). 'Used' means either registered by this method
     * or queried by the {@link #getAccountById(long)} method.
     */
     void putAccount(Account account);
}
```