Applying the Java ScheduledExecutor Service to TimedMemoizer

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

• Learn how to create a TimedMemoizer that applies ScheduledExecutorService to remove stale entries
Applying ScheduledExecutor Service to TimedMemoizer
Applying ScheduledExecutorService to TimedMemoizer

- TimedMemoizer maps a key to the value produced by a function, but limits the time a key/value pair remains cached.

See PrimeScheduledExecutorService/app/src/main/java/vandy/mooc/prime/utils/TimedMemoizer.java
Applying ScheduledExecutorService to TimedMemoizer

- TimedMemoizer maps a key to the value produced by a function, but limits the time a key/value pair remains cached.
- If a value has been computed for a key it is returned rather than calling the function to compute it again.
Applying ScheduledExecutorService to TimedMemoizer

- TimedMemoizer uses ConcurrentHashMap to minimize synchronization overhead

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html
Applying ScheduledExecutorService to TimedMemoizer

- TimedMemoizer uses ConcurrentHashMap to minimize synchronization overhead
- A different lock guards each hash bin

```
ConcurrentHashMap
```

```
0 1 2 n
Hash Bin Hash Bin Hash Bin Hash Bin
```

*Contention is low due to use of multiple locks*

Applying ScheduledExecutorService to TimedMemoizer

- TimedMemoizer uses ConcurrentHashMap to minimize synchronization overhead
  - A different lock guards each hash bin
  - A SynchronizedMap just uses one lock

**SynchronizedMap**

```
0 1 2 3 4  n
```

- key-value
- key-value
- key-value

Contention is higher due to use of one lock

See codepumpkin.com/hashtable-vs-synchronizedmap-vs-concurrenthashmap
Applying ScheduledExecutorService to TimedMemoizer

- TimedMemoizer uses ConcurrentHashMap to minimize synchronization overhead
  - A different lock guards each hash bin
- computeIfAbsent() ensures only one call to function runs when a key & value are first added to the cache

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html#computeIfAbsent
Applying ScheduledExecutorService to TimedMemoizer

- TimedMemoizer uses ConcurrentHashMap to minimize synchronization overhead
  - A different lock guards each hash bin
- `computeIfAbsent()` ensures only one call to function runs when a key & value are first added to the cache

Eliminates FutureTask (ashkrit.blogspot.com/2014/12/what-is-new-in-java8-concurrenthashmap.html)
Applying ScheduledExecutorService to TimedMemoizer

• If a key isn’t accessed within a given period, TimedMemoizer purges it from the map.
- If a key isn’t accessed within a given period, TimedMemoizer purges it from the map.
- RefCountedValue tracks the number of times a key is referenced within a given number of milliseconds.

Applying ScheduledExecutorService to TimedMemoizer:

```java
TimedMemoizer<K,V> mFunction: Function<K,V>
```

- `mTimeoutInMillisecs: long`
- `mScheduledExecutorService: ScheduledExecutorService`
- `TimedMemoizer(Function<K,V>,long)`, `apply(K)`, `shutdown():void`

```java
RefCountedValue mRefCount: AtomicLong
mValue: V
```

- `RefCountedValue(V,long)`, `equals(Object):boolean`, `get()`, `schedule(K):void`

-mNonAccessedValue 0..1
-mCache 0..*

12
Applying `ScheduledExecutorService` to `TimedMemoizer`

- If a key isn’t accessed within a given period, `TimedMemoizer` purges it from the map.
- `RefCountedValue` tracks the number of times a key is referenced within a given number of milliseconds.
- Timeout logic is performed by scheduling a new “removeIfStale” runnable via the Java `ScheduledExecutorService`.

Each runnable is scheduled as a “one-shot” task that’s rescheduled iff the value has been accessed during the `mTimeoutInMillisecs` period.
Applying ScheduledExecutorService to TimedMemoizer

- Lots of memory can be consumed with a large number of map entries since each key will create a new “removeIfStale” runnable.

See upcoming lesson on "Java ScheduledexecutorService: Application to TimedMemoizerEx"
End of Applying the Java ScheduledExecutorService to TimedMemoizer