Implementing TimedMemoizerEx with the Java ScheduledExecutorService

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

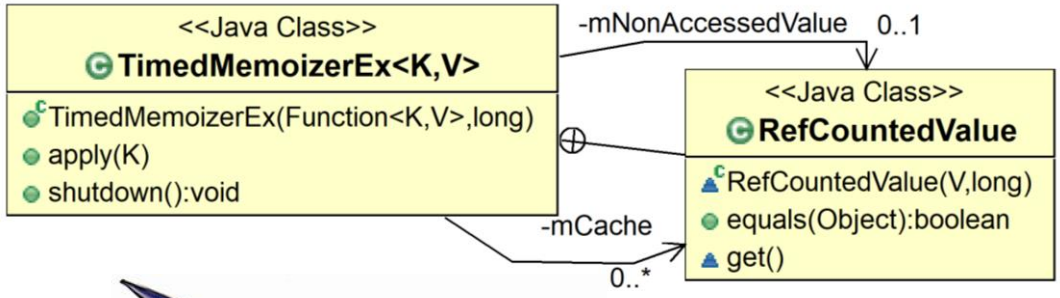
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Learn how to create a TimedMemoizerEx that applies the ScheduledExecutor Service to remove stale entries
- Know how to implement the TimedMemoizerEx class
  - This class applies the Java ScheduledExecutorService's periodic timer mechanism in a space efficient manner



Applying ScheduledExecutor Service to TimedMemoizerEx

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    final AtomicLong mRefCount;

    final V mValue;

    RefCountedValue(V value, long initCount)
    { mValue = value; mRefCount = new AtomicLong(initCount); }

    V get() {
      mRefCount.getAndIncrement();
      return mValue;
    } ...
```

See PrimeExecutorCompletionService/app/src/main/java/vandy/mooc/prime/utils/TimedMemoizerEx.java

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    final AtomicLong mRefCount;

    final V mValue;

    RefCountedValue(V value, long initCount)
    { mValue = value; mRefCount = new AtomicLong(initCount); }

    V get() {
      mRefCount.getAndIncrement();
      return mValue;
    } ...
```

TimedMemoizerEx can be used whenever a Function is expected

See docs.oracle.com/javase/8/docs/api/java/util/function/Function.html

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    final AtomicLong mRefCount;

    final V mValue;

    RefCountedValue(V value, long initCount)
    { mValue = value; mRefCount = new AtomicLong(initCount); }

    V get() {
      mRefCount.getAndIncrement();
      return mValue;
    } ...
```

Records the # of times a key is referenced within mTimeoutInMillisecs

Similar to the TimedMemoizer class, but omits the schedule() method

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    final AtomicLong mRefCount;

    final V mValue;

    RefCountedValue(V value, long initCount)
    { mValue = value; mRefCount = new AtomicLong(initCount); }

    V get() {
      mRefCount.getAndIncrement();
      return mValue;
    } ...
```

This value is incremented atomically every time that a key is accessed

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicLong.html

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    final AtomicLong mRefCount;

    final V mValue;

    RefCountedValue(V value, long initCount)
    { mValue = value; mRefCount = new AtomicLong(initCount); }

    V get() {
      mRefCount.getAndIncrement();
      return mValue;
    } ...
```

The value that's being reference counted

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    final AtomicLong mRefCount;

    final V mValue;

    RefCountedValue(V value, long initCount)
    { mValue = value; mRefCount = new AtomicLong(initCount); }

    V get() {
      mRefCount.getAndIncrement();
      return mValue;
    } ...
```

Constructor initializes the fields

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    final AtomicLong mRefCount;

    final V mValue;

    RefCountedValue(V value, long initCount)
    { mValue = value; mRefCount = new AtomicLong(initCount); }

    V get() {
      mRefCount.getAndIncrement();
      return mValue;
    } ...
```

Increment the ref count atomically & return the value

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    public boolean equals(Object obj) {
        if (getClass() != obj.getClass())
            return false;
        else {
            final RefCountedValue t = (RefCountedValue) obj;
            return mRefCount.get() == t.mRefCount.get();
        }
    }
    ...
```

Used by CHM.remove() to determine if a key has been accessed within mTimeoutInMillisecs

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    public boolean equals(Object obj) {
        if (getClass() != obj.getClass())
            return false;
        else {
            final RefCountedValue t = (RefCountedValue) obj;
            return mRefCount.get() == t.mRefCount.get();
        }
    }
    ...
```

Return true if ref counts are equal, else false

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  private final Map<K, RefCountedValue> mCache =
    new ConcurrentHashMap<>();
  private final long mTimeoutInMillisecs;
  private final Function<K, V> mFunction;
  private ScheduledExecutorService mSchedExecSvc;
  private final RefCountedValue mNonAccessedValue =
    new RefCountedValue(null, 1);
  private final ThresholdCrosser mCacheCount =
    new ThresholdCrosser(0);
  private ScheduledFuture<?> mScheduledFuture;
  ...
```

This map associates a key K with a value V that's produced by a function

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  private final Map<K, RefCountedValue> mCache =
    new ConcurrentHashMap<>();
```

Keeps track of the # of times a key/value is referenced within mTimeoutInMillisecs

```
  private final long mTimeoutInMillisecs;
  private final Function<K, V> mFunction;
  private ScheduledExecutorService mSchedExecSvc;
  private final RefCountedValue mNonAccessedValue =
    new RefCountedValue(null, 1);
  private final ThresholdCrosser mCacheCount =
    new ThresholdCrosser(0);
  private ScheduledFuture<?> mScheduledFuture;
  ...
```

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  private final Map<K, RefCountedValue> mCache =
    new ConcurrentHashMap<>();

  private final long mTimeoutInMillisecs;
  private final Function<K, V> mFunction;
  private ScheduledExecutorService mSchedExecSvc;
  private final RefCountedValue mNonAccessedValue =
    new RefCountedValue(null, 1);
  private final ThresholdCrosser mCacheCount =
    new ThresholdCrosser(0);
  private ScheduledFuture<?> mScheduledFuture;
  ...
```

The amount of time to retain a value in the cache

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  private final Map<K, RefCountedValue> mCache =
    new ConcurrentHashMap<>();

  private final long mTimeoutInMillisecs;
  private final Function<K, V> mFunction;
  private ScheduledExecutorService mSchedExecSvc;
  private final RefCountedValue mNonAccessedValue =
    new RefCountedValue(null, 1);
  private final ThresholdCrosser mCacheCount =
    new ThresholdCrosser(0);
  private ScheduledFuture<?> mScheduledFuture;
  ...
```

This function produces a value based on a key

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  private final Map<K, RefCountedValue> mCache =
    new ConcurrentHashMap<>();

  private final long mTimeoutInMillisecs;
  private final Function<K, V> mFunction;
  private ScheduledExecutorService mSchedExecSvc;
  private final RefCountedValue mNonAccessedValue =
    new RefCountedValue(null, 1);
  private final ThresholdCrosser mCacheCount =
    new ThresholdCrosser(0);
  private ScheduledFuture<?> mScheduledFuture;
  ...
```

Executes one runnable periodically to check for & remove expired keys

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  private final Map<K, RefCountedValue> mCache =
    new ConcurrentHashMap<>();
```

A ref count of 1 is used to check if a key's not been accessed in mTimeoutInMillisecs

```
  private final long mTimeoutInMillisecs;
  private final Function<K, V> mFunction;
  private ScheduledExecutorService mSchedExecSvc;
  private final RefCountedValue mNonAccessedValue =
    new RefCountedValue(null, 1);
  private final ThresholdCrosser mCacheCount =
    new ThresholdCrosser(0);
  private ScheduledFuture<?> mScheduledFuture;
  ...
```

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  private final Map<K, RefCountedValue> mCache =
    new ConcurrentHashMap<>();


  private final long mTimeoutInMillisecs;
  private final Function<K, V> mFunction;
  private ScheduledExecutorService mSchedExecSvc;
  private final RefCountedValue mNonAccessedValue =
    new RefCountedValue(null, 1);
  private final ThresholdCrosser mCacheCount =
    new ThresholdCrosser(0);
  private ScheduledFuture<?> mScheduledFuture;
  ...
```

Tracks the # of entries in mCache

This object ensures mPurgeEntries is properly scheduled & cancelled

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  private final Map<K, RefCountedValue> mCache =
    new ConcurrentHashMap<>();


  private final long mTimeoutInMillisecs;
  private final Function<K, V> mFunction;
  private ScheduledExecutorService mSchedExecSvc;
  private final RefCountedValue mNonAccessedValue =
    new RefCountedValue(null, 1);
  private final ThresholdCrosser mCacheCount =
    new ThresholdCrosser(0);
  private ScheduledFuture<?> mScheduledFuture;
  ...
```

Used to cancel mPurgeEntries

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private final Runnable mPurgeEntries = () -> {
    mCache.forEach((key, value) -> {
      long oldCount =
        value.mRefCount.get();
      if (mCache.remove(key, mNonAccessedValue))
        mCacheCount.decrementAndCallAtN(0,
                                        () -> mScheduledFuture.cancel(true));

      else
        value.mRefCount.getAndUpdate
          (curCount -> curCount > oldCount ? curCount : 1);
    }}; ...
```

Periodically purges stale map entries

apply() schedules this runnable whenever the 1st entry is added to the cache

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private final Runnable mPurgeEntries = () -> {
    mCache.forEach((key, value) -> {
      long oldCount =
        value.mRefCount.get();
      if (mCache.remove(key, mNonAccessedValue))
        mCacheCount.decrementAndCallAtN(0,
                                        () -> mScheduledFuture.cancel(true));

      else
        value.mRefCount.getAndUpdate
          (curCount -> curCount > oldCount ? curCount : 1);
    }}; ...
```

Iterate through the map purging stale entries

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private final Runnable mPurgeEntries = () -> {
    mCache.forEach((key, value) -> {
      long oldCount =
        value.mRefCount.get();
      if (mCache.remove(key, mNonAccessedValue))
        mCacheCount.decrementAndCallAtN(0,
                                        () -> mScheduledFuture.cancel(true));

      else
        value.mRefCount.getAndUpdate
          (curCount -> curCount > oldCount ? curCount : 1);
    }}; ...
```

Store this value's ref count

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private final Runnable mPurgeEntries = () -> {
    mCache.forEach((key, value) -> {
      long oldCount =
        value.mRefCount.get();
      if (mCache.remove(key, mNonAccessedValue))
        mCacheCount.decrementAndCallAtN(0,
                                        () -> mScheduledFuture.cancel(true));

      else
        value.mRefCount.getAndUpdate
          (curCount -> curCount > oldCount ? curCount : 1);
    }}; ...
```

Atomically remove stale entry

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private final Runnable mPurgeEntries = () -> {
    mCache.forEach((key, value) -> {
      long oldCount =
        value.mRefCount.get();
      if (mCache.remove(key, mNonAccessedValue))
        mCacheCount.decrementAndCallAtN(0,
                                        () -> mScheduledFuture.cancel(true));
      else
        value.mRefCount.getAndUpdate
          (curCount -> curCount > oldCount ? curCount : 1);
    }}; ...
```

Atomically stop purging entries when the map is empty

There's no point in trying to purge the entries in an empty map!!!!

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private final Runnable mPurgeEntries = () -> {
    mCache.forEach((key, value) -> {
      long oldCount =
        value.mRefCount.get();
      if (mCache.remove(key, mNonAccessedValue))
        mCacheCount.decrementAndCallAtN(0,
                                        () -> mScheduledFuture.cancel(true));
      else
        value.mRefCount.getAndUpdate
          (curCount -> curCount > oldCount ? curCount : 1);
    }}; ...
```

Try resetting ref count to 1 so it won't be considered as accessed

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicLong.html#getAndUpdate

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  private final Runnable mPurgeEntries = () -> {
    mCache.forEach((key, value) -> {
      long oldCount =
        value.mRefCount.get();
      if (mCache.remove(key, mNonAccessedValue))
        mCacheCount.decrementAndCallAtN(0,
                                () -> mScheduledFuture.cancel(true));

      else
        value.mRefCount.getAndUpdate
          (curCount -> curCount > oldCount ? curCount : 1);
    }}; ...
```

Don't reset ref count if it's increased between remove() & here

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> { mCacheCount.incrementAndCallAtN(1, () -> {
        mScheduledFuture = mSchedExecSvc.scheduleAtFixedDelay
          (mPurgeEntries, mTimeoutInMillisecs,
           mTimeoutInMillisecs, MILLISECONDS); });

        return new RefCountedValue(mFunction.apply(k), 0);
      });
    return rcValue.get();
  } ...
```

Return value associated with key in cache

apply() schedules the one runnable created for all key/value pairs in the map

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> { mCacheCount.incrementAndCallAtN(1, () -> {
        mScheduledFuture = mSchedExecSvc.scheduleAtFixedDelay
          (mPurgeEntries, mTimeoutInMillisecs,
           mTimeoutInMillisecs, MILLISECONDS); });


        return new RefCountedValue(mFunction.apply(k), 0);
      });
    return rcValue.get();
  } ...
```

Atomic "check-then-act" method

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

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> { mCacheCount.incrementAndCallAtN(1, () -> {
        mScheduledFuture = mSchedExecSvc.scheduleAtFixedDelay
          (mPurgeEntries, mTimeoutInMillisecs,
           mTimeoutInMillisecs, MILLISECONDS); });
        return new RefCountedValue(mFunction.apply(k), 0);
      });
    return rcValue.get();
  } ...
```

If value already computed for key then just return it

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> { mCacheCount.incrementAndCallAtN(1, () -> {
        mScheduledFuture = mSchedExecSvc.scheduleAtFixedDelay
          (mPurgeEntries, mTimeoutInMillisecs,
           mTimeoutInMillisecs, MILLISECONDS); });
        return new RefCountedValue(mFunction.apply(k), 0);
      });
    return rcValue.get();
  } ...
```

Otherwise, atomically run this lambda to create, schedule, & return a new value

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> { mCacheCount.incrementAndCallAtN(1, () -> {
        mScheduledFuture = mSchedExecSvc.scheduleAtFixedDelay
          (mPurgeEntries, mTimeoutInMillisecs,
           mTimeoutInMillisecs, MILLISECONDS); });
        return new RefCountedValue(mFunction.apply(k), 0);
      });
    return rcValue.get();
  } ...
```

Atomically schedule mPurgeEntries to run periodically when 1st entry added to cache

We'll show the implementation of incrementAndCallAtN() shortly

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> { mCacheCount.incrementAndCallAtN(1, () -> {
        mScheduledFuture = mSchedExecSvc.scheduleAtFixedDelay
          (mPurgeEntries, mTimeoutInMillisecs,
           mTimeoutInMillisecs, MILLISECONDS); });
        return new RefCountedValue(mFunction.apply(k), 0);
      });
    return rcValue.get();
  } ...
```

mPurgeEntries will purge keys that aren't accessed within mTimeoutInMillisecs

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledExecutorService.html#scheduleWithFixedDelay

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> { mCacheCount.incrementAndCallAtN(1, () -> {
        mScheduledFuture = mSchedExecSvc.scheduleAtFixedDelay
          (mPurgeEntries, mTimeoutInMillisecs,
           mTimeoutInMillisecs, MILLISECONDS); });
        return new RefCountedValue(mFunction.apply(k), 0);
      });
    return rcValue.get();
  } ...
```

Result is stored in a ScheduledFuture so it can be cancelled when necessary.

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledFuture.html

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> { mCacheCount.incrementAndCallAtN(1, () -> {
        mScheduledFuture = mSchedExecSvc.scheduleAtFixedDelay
          (mPurgeEntries, mTimeoutInMillisecs,
           mTimeoutInMillisecs, MILLISECONDS); });

        return new RefCountedValue(mFunction.apply(k), 0);
      });
    return rcValue.get();
  } ...
```

Compute the function, store its result in a ref-counted value (initially 0), & return this value

mFunction.apply() runs in the thread of control of the method that called apply()

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> { mCacheCount.incrementAndCallAtN(1, () -> {
        mScheduledFuture = mSchedExecSvc.scheduleAtFixedDelay
          (mPurgeEntries, mTimeoutInMillisecs,
           mTimeoutInMillisecs, MILLISECONDS); });

        return new RefCountedValue(mFunction.apply(k), 0);
      });
    return rcValue.get();
  } ...
```

This object is value associated with key

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> { mCacheCount.incrementAndCallAtN(1, () -> {
        mScheduledFuture = mSchedExecSvc.scheduleAtFixedDelay
          (mPurgeEntries, mTimeoutInMillisecs,
           mTimeoutInMillisecs, MILLISECONDS); });


        return new RefCountedValue(mFunction.apply(k), 0);
      });
    return rcValue.get();
  } ...
```

Increment the reference count atomically & return the value

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public TimedMemoizerEx(Function<K, V> function, long timeout){
    mFunction = function;
    mTimeoutInMillisecs = timeout;
    mSchedExecSvc =
      Executors.newScheduledThreadPool(1);

    ScheduledThreadPoolExecutor exec =
        (ScheduledThreadPoolExecutor) mSchedExecSvc;
    exec.setRemoveOnCancelPolicy(true);
    exec.setContinueExistingPeriodicTasksAfterShutdownPolicy(false);
    exec.setExecuteExistingDelayedTasksAfterShutdownPolicy(false);
  }...
```

Constructor initializes the fields

The constructor is the same as in the TimedMemoizer class

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public TimedMemoizerEx(Function<K, V> function, long timeout){
    mFunction = function;
    mTimeoutInMillisecs = timeout;
    mSchedExecSvc =
      Executors.newScheduledThreadPool(1);

    ScheduledThreadPoolExecutor exec =
        (ScheduledThreadPoolExecutor) mSchedExecSvc;
    exec.setRemoveOnCancelPolicy(true);
    exec.setContinueExistingPeriodicTasksAfterShutdownPolicy(false);
    exec.setExecuteExistingDelayedTasksAfterShutdownPolicy(false);
  }...
```

Store function & timeout params

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public TimedMemoizerEx(Function<K, V> function, long timeout){
    mFunction = function;
    mTimeoutInMillisecs = timeout;
    mSchedExecSvc =
      Executors.newScheduledThreadPool(1);

    ScheduledThreadPoolExecutor exec =
        (ScheduledThreadPoolExecutor) mSchedExecSvc;
    exec.setRemoveOnCancelPolicy(true);
    exec.setContinueExistingPeriodicTasksAfterShutdownPolicy(false);
    exec.setExecuteExistingDelayedTasksAfterShutdownPolicy(false);
  }...
```

Create a ScheduledThreadPool Executor containing one thread

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public TimedMemoizerEx(Function<K, V> function, long timeout){
    mFunction = function;
    mTimeoutInMillisecs = timeout;
    mSchedExecSvc =
      Executors.newScheduledThreadPool(1);

    ScheduledThreadPoolExecutor exec =
        (ScheduledThreadPoolExecutor) mSchedExecSvc;
    exec.setRemoveOnCancelPolicy(true);
    exec.setContinueExistingPeriodicTasksAfterShutdownPolicy(false);
    exec.setExecuteExistingDelayedTasksAfterShutdownPolicy(false);
  }...
```

Get an object to set policies that clean everything up on shutdown

Implementing TimedMemoizerEx w/ScheduledExecutorService

- TimedMemoizerEx uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizerEx<K, V> implements Function<K, V> {
  ...
  public TimedMemoizerEx(Function<K, V> function, long timeout){
    mFunction = function;
    mTimeoutInMillisecs = timeout;
    mSchedExecSvc =
      Executors.newScheduledThreadPool(1);

    ScheduledThreadPoolExecutor exec =
        (ScheduledThreadPoolExecutor) mSchedExecSvc;
    exec.setRemoveOnCancelPolicy(true);
    exec.setContinueExistingPeriodicTasksAfterShutdownPolicy(false);
    exec.setExecuteExistingDelayedTasksAfterShutdownPolicy(false);
  }...
```

Set shutdown policies that remove scheduled runnables & disable tasks

See stackoverflow.com/questions/3893609/scheduledthreadpoolexecutor-setxxxxshutdownpolicy

Implementing TimedMemoizerEx w/ScheduledExecutorService

- ThreadholdCrosser atomically increments & decrements an internal count, calling a given action when the internal count equals a given parameter

```
class ThresholdCrosser {
  private int mCount;

  public ThresholdCrosser(int initCount) { mCount = initCount; }

  public synchronized void incrementAndCallAtN(int n,
                                               Runnable action)
  { if (++mCount == n) action.run(); }


  public synchronized void decrementAndCallAtN(int n,
                                               Runnable action)
  { if (--mCount == n) action.run(); } ...
```

See PrimeExecutorCompletionService/app/src/main/java/vandy/mooc/prime/utils/ThresholdCrosser.java

Implementing TimedMemoizerEx w/ScheduledExecutorService

- ThreadholdCrosser atomically increments & decrements an internal count, calling a given action when the internal count equals a given parameter

```
class ThresholdCrosser {
  private int mCount;

  public ThresholdCrosser(int initCount) { mCount = initCount; }

  public synchronized void incrementAndCallAtN(int n,
                                               Runnable action)
  { if (++mCount == n) action.run(); }


  public synchronized void decrementAndCallAtN(int n,
                                               Runnable action)
  { if (--mCount == n) action.run(); } ...
```

Constructor sets the field

Implementing TimedMemoizerEx w/ScheduledExecutorService

- ThreadholdCrosser atomically increments & decrements an internal count, calling a given action when the internal count equals a given parameter

```
class ThresholdCrosser {
  private int mCount;

  public ThresholdCrosser(int initCount) { mCount = initCount; }

  public synchronized void incrementAndCallAtN(int n,
                                               Runnable action)
  { if (++mCount == n) action.run(); }

  public synchronized void decrementAndCallAtN(int n,
                                               Runnable action)
  { if (--mCount == n) action.run(); } ...
```

Invoke action iff internal count equals n after being incremented

TimedMemoizerEx uses this method to (re)schedule the mPurgedEntries task

Implementing TimedMemoizerEx w/ScheduledExecutorService

- ThreadholdCrosser atomically increments & decrements an internal count, calling a given action when the internal count equals a given parameter

```
class ThresholdCrosser {
  private int mCount;

  public ThresholdCrosser(int initCount) { mCount = initCount; }

  public synchronized void incrementAndCallAtN(int n,
                                               Runnable action)
  { if (++mCount == n) action.run(); }

  public synchronized void decrementAndCallAtN(int n,
                                               Runnable action)
  { if (--mCount == n) action.run(); } ...
```

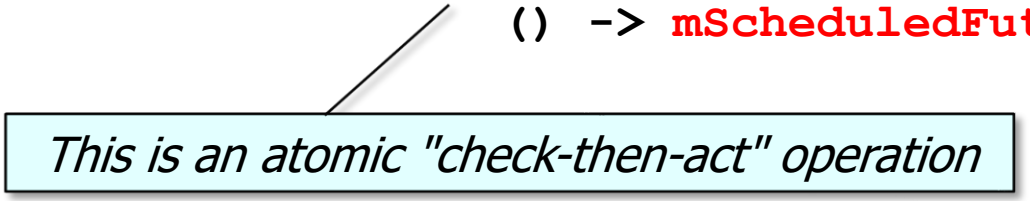
Invoke action iff internal count equals n after being decremented

TimedMemoizerEx uses this method to cancel the mPurgedEntries task

Implementing TimedMemoizerEx w/ScheduledExecutorService

- ThreadholdCrosser is needed to avoid a race condition between testing if mCache.size() == 0 & then starting/stopping the scheduled future

```
if (mCache.remove(key, mNonAccessedValue))
  mCacheCount
    .decrementAndCallAtN(0,
                         () -> mScheduledFuture.cancel(true));
```



versus

```
if (mCache.remove(key, mNonAccessedValue) {
  if (mCache.size() == 0)
    mScheduledFuture.cancel(true);
```

Implementing TimedMemoizerEx w/ScheduledExecutorService

- ThreadholdCrosser is needed to avoid a race condition between testing if mCache.size() == 0 & then starting/stopping the scheduled future

```
if (mCache.remove(key, mNonAccessedValue))
  mCacheCount
    .decrementAndCallAtN(0,
                         () -> mScheduledFuture.cancel(true));
```

versus

```
if (mCache.remove(key, mNonAccessedValue) {
  if (mCache.size() == 0)
    mScheduledFuture.cancel(true);
```

Race condition since this is not an atomic "check-then-act" operation

See dig.cs.illinois.edu/papers/checkThenAct.pdf

End of Implementing Timed MemoizerEx with the Java ScheduledExecutorService