Implementing TimedMemoizer with the Java ScheduledExecutorService

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

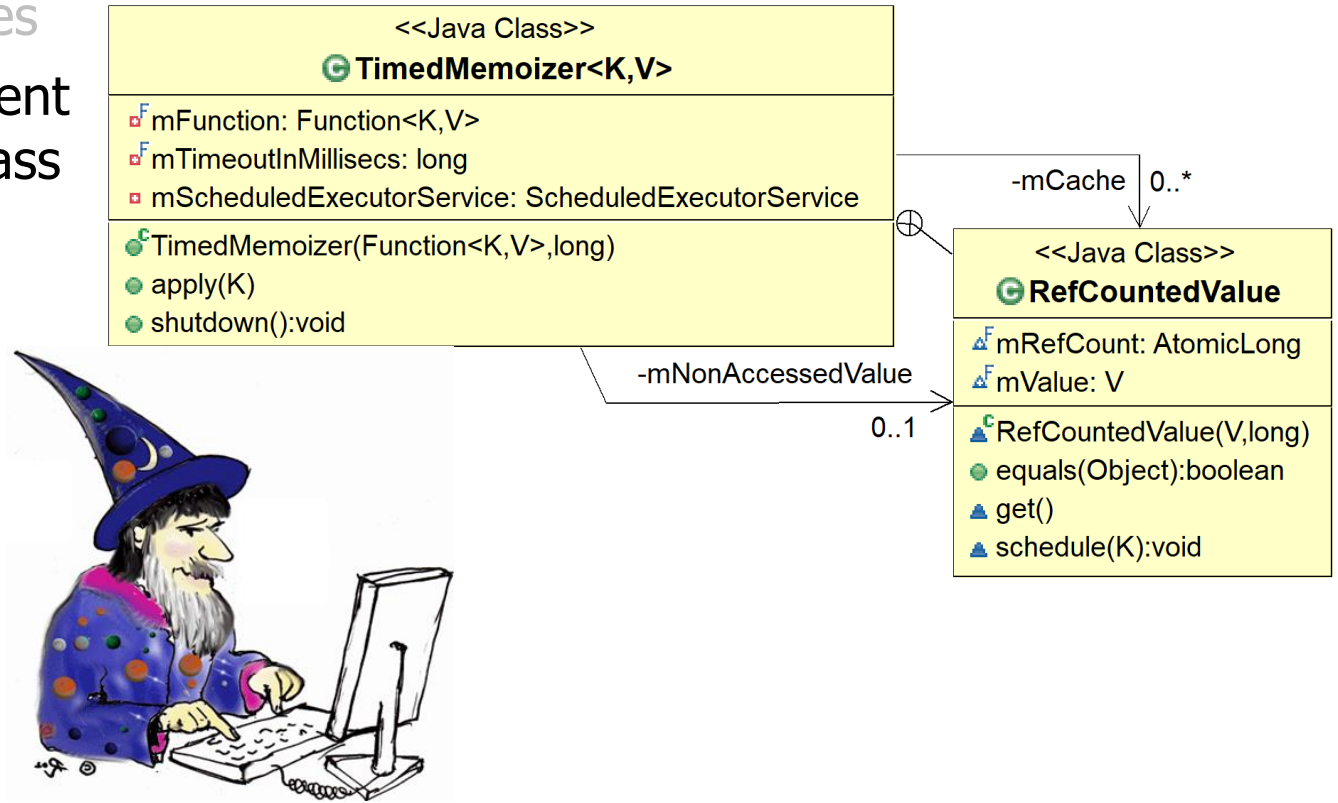
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Learn how to create a TimedMemoizer that applies ScheduledExecutorService to remove stale entries
- Know how to implement the TimeMemoizer class
  - This class uses the ScheduledExecutor Service to remove stale entries



Implementing TimedMemoizer w/ScheduledExecutorService

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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);
  ...
```

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

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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 final ScheduledExecutorService mSchedExecSvc;

  private final RefCountedValue mNonAccessedValue =
    new RefCountedValue(null, 1);
  ...
```

TimedMemoizer can be used transparently whenever a Function is expected

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

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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);
  ...
```

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 TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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);
  ...
```

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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);
  ...
```

The amount of time to retain a value in the cache

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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);
  ...
```

This function produces a value based on the key

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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);
  ...
```

Executes runnables after timeouts elapse to check for & remove expired keys

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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);
  ...
```

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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 TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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 TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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 TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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 TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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 TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<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 TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    void schedule(K key) {
      Runnable removeIfStale = new Runnable() {
        public void run() {
          long oldCount = mRefCount.get();
          if (mCache.remove(key, mNonAccessedValue)) { }
          else {
            ...
          }
      }; ...
```

Schedule runnable that removes key from cache if it is "stale"

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    void schedule(K key) {
      Runnable removeIfStale = new Runnable() {
        public void run() {
          long oldCount = mRefCount.get();
          if (mCache.remove(key, mNonAccessedValue)) { }
          else {
            ...
          }
      }; ...
```

Runnable checks if cached entry became "stale" & remove it if so

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    void schedule(K key) {
      Runnable removeIfStale = new Runnable() {
        public void run() {
          long oldCount = mRefCount.get();
          if (mCache.remove(key, mNonAccessedValue)) { }
          else {
            ...
          }
      }; ...
```

The ScheduledExecutorService calls this hook method when time elapses

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    void schedule(K key) {
      Runnable removeIfStale = new Runnable() {
        public void run() {
          long oldCount = mRefCount.get();
          if (mCache.remove(key, mNonAccessedValue)) { }
          else {
            ... mSchedExecSvc.schedule(this, ...);
          }
      }; ...
```

Can't use a lambda since we reference "this" in run()!

See stackoverflow.com/questions/24202236/lambda-this-reference-in-java/24202272#24202272

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    void schedule(K key) {
      Runnable removeIfStale = new Runnable() {
        public void run() {
          long oldCount = mRefCount.get();
          if (mCache.remove(key, mNonAccessedValue)) { }
          else {
            ...
          }
      }; ...
```

Store current ref count & begin "staleness" check

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    void schedule(K key) {
      Runnable removeIfStale = new Runnable() {
        public void run() {
          long oldCount = mRefCount.get();
          if (mCache.remove(key, mNonAccessedValue)) { }
          else {
            ...
          }
      }; ...
```

Remove key only if it's not been accessed in mTimeoutInMillisecs

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    void schedule(K key) {
      Runnable removeIfStale = new Runnable() {
        public void run() {
          ... else {
            mRefCount.getAndUpdate
              (curCount -> curCount > oldCount ? curCount : 1);
             mSchedExecSvc.schedule
              (this, mTimeoutInMillisecs, MILLISECONDS);
          }
      }; ...
```

Try to atomically reset ref count to 1 so it won't be considered accessed

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

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    void schedule(K key) {
      Runnable removeIfStale = new Runnable() {
        public void run() {
          ... else {
            mRefCount.getAndUpdate
               (curCount -> curCount > oldCount ? curCount : 1);
             mSchedExecSvc.schedule
               (this, mTimeoutInMillisecs, MILLISECONDS);
          }
      }; ...
```

Do NOT reset it to 1 if ref count has increased concurrently

Ref count can increase concurrently since mCache is never locked entirely!

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    void schedule(K key) {
      Runnable removeIfStale = new Runnable() {
        public void run() {
          ... else {
            mRefCount.getAndUpdate
               (curCount -> curCount > oldCount ? curCount : 1);
             mSchedExecSvc.schedule
               (this, mTimeoutInMillisecs, MILLISECONDS);
          }
      }; ...
```

Reschedule this runnable to run again in mTimeoutInMillisecs

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

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  private class RefCountedValue {
    ...
    void schedule(K key) {
      Runnable removeIfStale = new Runnable() {
        ...
      };

      mSchedExecSvc.schedule
            (removeIfStale, mTimeoutInMillisecs, MILLISECONDS);
    }
}
```

Initially schedule this runnable to execute after mTimeoutInMillisecs

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public TimedMemoizer(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

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public TimedMemoizer(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 TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public TimedMemoizer(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 TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public TimedMemoizer(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 TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public TimedMemoizer(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 TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> {
          RefCountedValue rcv =
            new RefCountedValue(mFunction.apply(k), 0);
          if (!Thread.currentThread().isInterrupted()
              && mTimeoutInMillisecs > 0)
            rcv.schedule(key);
          return rcv;
        });
    return rcValue.get();
  } ...
```

Return value associated with key in cache

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> {
          RefCountedValue rcv =
            new RefCountedValue(mFunction.apply(k), 0);
          if (!Thread.currentThread().isInterrupted()
              && mTimeoutInMillisecs > 0)
            rcv.schedule(key);
          return rcv;
        });
    return rcValue.get();
  } ...
```

Atomic "check-then-act" method

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

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> {
          RefCountedValue rcv =
            new RefCountedValue(mFunction.apply(k), 0);
          if (!Thread.currentThread().isInterrupted()
              && mTimeoutInMillisecs > 0)
            rcv.schedule(key);
          return rcv;
       });
    return rcValue.get();
  } ...
```

If value already computed for key then just return it

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> {
          RefCountedValue rcv =
            new RefCountedValue(mFunction.apply(k), 0);
          if (!Thread.currentThread().isInterrupted()
              && mTimeoutInMillisecs > 0)
            rcv.schedule(key);
          return rcv;
        });
    return rcValue.get();
  } ...
```

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

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> {
          RefCountedValue rcv =
            new RefCountedValue(mFunction.apply(k), 0);
          if (!Thread.currentThread().isInterrupted()
              && mTimeoutInMillisecs > 0)
            rcv.schedule(key);
          return rcv;
        });
    return rcValue.get();
  } ...
```

Compute the function & store its result in a reference-counted value (initially 0)

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

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> {
          RefCountedValue rcv =
            new RefCountedValue(mFunction.apply(k), 0);
          if (!Thread.currentThread().isInterrupted()
              && mTimeoutInMillisecs > 0)
            rcv.schedule(key);
          return rcv;
        });
    return rcValue.get();
  } ...
```

Schedule this object/key to run after mTimeoutInMillisecs

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> {
          RefCountedValue rcv =
            new RefCountedValue(mFunc
          if (!Thread.currentThread()
              && mTimeoutInMillisecs
            rcv.schedule(key);
          return rcv;
        });
    return rcValue.get();
  } ...
```

A runnable is created for each key/value pair that it passes to ScheduledExecutorService

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> {
          RefCountedValue rcv =
            new RefCountedValue(mFunction.apply(k), 0);
          if (!Thread.currentThread().isInterrupted()
              && mTimeoutInMillisecs > 0)
            rcv.schedule(key);
          return rcv;
      });
    return rcValue.get();
  } ...
```

Return this object as the value associated with the key

Implementing TimedMemoizer w/ScheduledExecutorService

- The TimedMemoizer uses ScheduledExecutorService to remove stale entries

```
class TimedMemoizer<K, V> implements Function<K, V> {
  ...
  public V apply(K key) {
    RefCountedValue rcValue = mCache.computeIfAbsent
      (key, (k) -> {
          RefCountedValue rcv =
            new RefCountedValue(mFunction.apply(k), 0);
          if (!Thread.currentThread().isInterrupted()
              && mTimeoutInMillisecs > 0)
            rcv.schedule(key);
          return rcv;
        });
    return rcValue.get();
  } ...
```

Increment the ref count atomically & return the value

End of Implementing Timed Memoizer with the Java ScheduledExecutorService