### The Java FutureTask: Applying Memoizer to the PrimeChecker App



Douglas C. Schmidt

<u>d.schmidt@vanderbilt.edu</u>

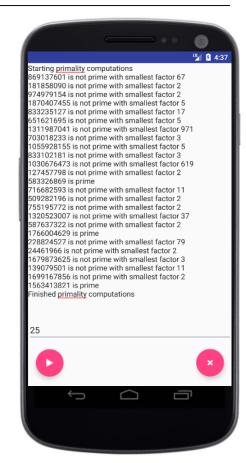
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems Vanderbilt University Nashville, Tennessee, USA

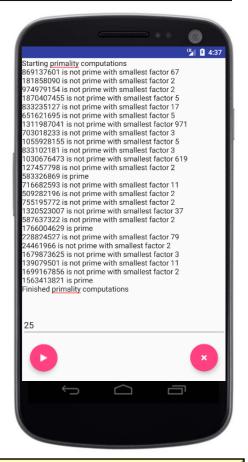


#### Learning Objectives in this Part of the Lesson

- Understand how Java FutureTask conveys a result from a computation running in a thread to thread(s) retrieving the result
- Recognize key methods in Java FutureTask
- Know what the Memoizer class is & why it uses FutureTask to optimize programs
- Learn how to implement the Memoizer with FutureTask
- Recognize how the Memoizer class is applied to the PrimeChecker app to optimize prime # checking

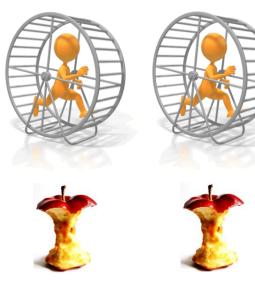


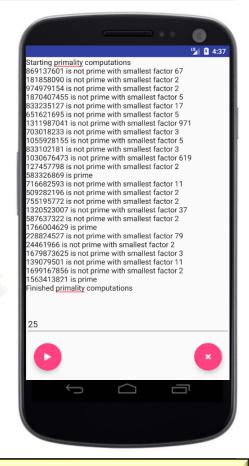
 This app applies the Java ExecutorCompletionService framework & the FutureTask-based Memoizer to check if N random #'s are prime



- This app applies the Java ExecutorCompletionService framework & the FutureTask-based Memoizer to check if N random #'s are prime
  - This app is "embarrassingly parallel" & compute-bound







See en.wikipedia.org/wiki/Embarrassingly\_parallel & en.wikipedia.org/wiki/CPU-bound

<<Java Class>> MainActivity checks primality of "count" random #'s via an **⊚** MainActivity ExecutorService w/a thread pool & the PrimeCallable class onCreate(Bundle):void onRetainNonConfigurationInstance():Object initializeViews():void setCount(View):void <<Java Class>> startOrStopComputations(View):void PrimeCallable startComputations(int):void interruptComputations():void odone():void o call():PrimeResult println(String):void onDestroy():void ~mActivity 0..1 -mRetainedState <<Java Class>> RetainedState <<Java Class>> PrimeResult △ mExecutorService: ExecutorService △ mThread: Thread PrimeResult(long,long) RetainedState() shutdown():void ~mFutureRunnable 0..1 <<Java Class>> FutureRunnable FutureRunnable(MainActivity,List<Future<PrimeResult>>) ▲ setActivity(MainActivity):void

See PrimeExecutorServiceFutureTask/app/src/main/java/vandy/mooc/prime/activities/MainActivity.java

o run():void

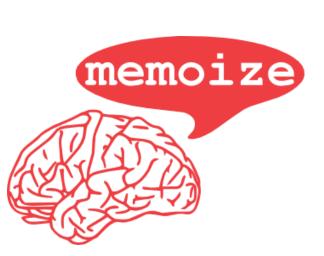
<<Java Class>> MainActivity checks primality of "count" random #'s via an **⊚** MainActivity ExecutorService w/a thread pool & the PrimeCallable class onCreate(Bundle):void onRetainNonConfigurationInstance():Object initializeViews():void setCount(View):void <<Java Class>> startOrStopComputations(View):void PrimeCallable startComputations(int):void Fixed-sized PrimeCallable(long,Function<Long,Long>) interruptComputations():void Thread Pool o done():void call():PrimeResult println(String):void A pool of worker threads onDestroy():void ~mActivity 0...1 -mRetainedState mExecutorService = Executors <<Java Class>> .newFixedThreadPool RetainedState <<Java Class>> PrimeResult △ mExecutorService: ExecutorService (Runtime △ mThread: Thread PrimeResult(long,long) .getRuntime() RetainedState() shutdown():void .availableProcessors()); ~mFutureRunnable 0..1 <<Java Class>> FutureRunnable The executor service uses a fixed-sized thread pool FutureRunnable(MainActivity,List<Future<PrimeResult>>) setActivity(MainActivity):void o run():void

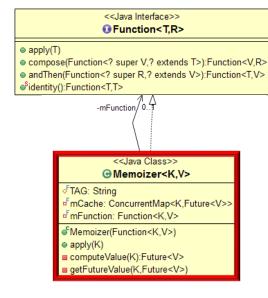
See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newFixedThreadPool

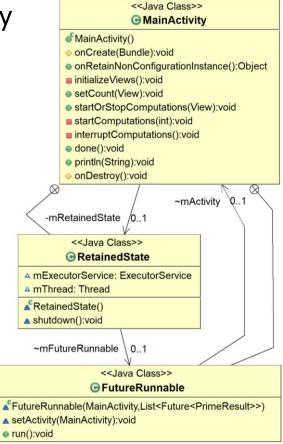
<<Java Class>> MainActivity checks primality of "count" random #'s via an **⊚** MainActivity ExecutorService w/a thread pool & the PrimeCallable class onCreate(Bundle):void onRetainNonConfigurationInstance():Object initializeViews():void setCount(View):void <<Java Class>> startOrStopComputations(View):void PrimeCallable startComputations(int):void Fixed-sized PrimeCallable(long,Function<Long,Long>) interruptComputations():void Thread Pool o done():void call():PrimeResult A pool of worker threads println(String):void onDestroy():void ~mActivity 0...1 -mRetainedState mExecutorService = Executors <<Java Class>> .newFixedThreadPool RetainedState <<Java Class>> PrimeResult △ mExecutorService: ExecutorService (Runtime △ mThread: Thread PrimeResult(long,long) .getRuntime() RetainedState() shutdown():void .availableProcessors()); ~mFutureRunnable 0..1 <<Java Class>> FutureRunnable Pool size tuned to # of processor cores FutureRunnable(MainActivity,List<Future<PrimeResult>>) setActivity(MainActivity):void o run():void

See docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#availableProcessors

 MainActivity also uses a memoizer to optimize primality checking of the random #'s







See earlier parts of this lesson on "Application to Memoizer" & "Implementing a Memoizer"

PrimeCallable uses a Function object to extensibly determine if a # is prime

```
class PrimeCallable
                                                           <<Java Class>>
                                                           PrimeCallable
       implements Callable<PrimeResult> {
                                                    PrimeCallable(long,Function<Long,Long>)
  long mPrimeCandidate;
                                                    call():PrimeResult
  mFunction<Long, Long> mPrimeChecker;
  PrimeCallable (Long primeCandidate,
                   Function<Long, Long>
                                                           <<Java Class>>
                       primeChecker) {
                                                           PrimeResult
    mPrimeCandidate = primeCandidate;
                                                         PrimeResult(long,long)
    mPrimeChecker = primeChecker;
```

PrimeCallable uses a Function object to extensibly determine if a # is prime

```
class PrimeCallable
                                                            <<Java Class>>
                                                           PrimeCallable
       implements Callable<PrimeResult> {
                                                    PrimeCallable(long,Function<Long,Long>)
  long mPrimeCandidate;
                                                    call():PrimeResult
  mFunction<Long, Long> mPrimeChecker;
  PrimeCallable (Long primeCandidate,
                   Function<Long, Long>
                                                            <<Java Class>>
                       primeChecker) {
                                                           PrimeResult
    mPrimeCandidate = primeCandidate;

▲ PrimeResult(long,long)

    mPrimeChecker = primeChecker;
```

PrimeCallable implements Callable so it can be submitted & run by the Java ExecutorCompletionService framework

PrimeCallable uses a Function object to extensibly determine if a # is prime

```
class PrimeCallable
                                                           <<Java Class>>
                                                           PrimeCallable
       implements Callable<PrimeResult> {
                                                    PrimeCallable(long,Function<Long,Long>)
  long mPrimeCandidate;
                                                    call():PrimeResult
  mFunction<Long, Long> mPrimeChecker;
  PrimeCallable (Long primeCandidate,
                   Function<Long, Long>
                                                           <<Java Class>>
                       primeChecker) {
                                                           PrimeResult
    mPrimeCandidate = primeCandidate;
                                                         PrimeResult(long,long)
    mPrimeChecker = primeChecker;
```

The function that checks primes is passed as a param & stored in a field

PrimeCallable uses a Function object to extensibly determine if a # is prime

```
class PrimeCallable
                                                                <<Java Class>>
                                                               PrimeCallable
       implements Callable<PrimeResult> {
                                                        PrimeCallable(long,Function<Long,Long>)
         The call() hook method applies the function
                                                        call():PrimeResult
  PrimeResult call()
     return new PrimeResult
        (mPrimeCandidate,
                                                                <<Java Class>>
         mPrimeChecker
                                                               PrimeResult
            .apply(mPrimeCandidate));
                                                             PrimeResult(long,long)
```

PrimeCallable uses a Function object to extensibly determine if a # is prime

```
class PrimeCallable
                                                                 <<Java Class>>
                                                                PrimeCallable
        implements Callable<PrimeResult> {
                                                        PrimeCallable(long,Function<Long,Long>)
                                                        call():PrimeResult
  PrimeResult call() {
     return new PrimeResult
        (mPrimeCandidate,
                                                                <<Java Class>>
         mPrimeChecker
                                                                PrimeResult
            .apply(mPrimeCandidate));
                                                              PrimeResult(long,long)
    apply() returns 0 if the # is prime or smallest factor if it's not
```

The apply() method call can be transparently optimized via the Memoizer

PrimeCallable uses a Function object to extensibly determine if a # is prime

```
class PrimeCallable
                                                                <<Java Class>>
                                                                PrimeCallable
        implements Callable<PrimeResult> {
                                                        PrimeCallable(long,Function<Long,Long>)
                                                        call():PrimeResult
  PrimeResult call() {
     return new PrimeResult
        (mPrimeCandidate,
                                                                <<Java Class>>
         mPrimeChecker
                                                                PrimeResult
            .apply (mPrimeCandidate) );
                                                              PrimeResult(long,long)
           The PrimeResult tuple matches the prime #
          candidate with result of checking for primality
```

```
<<Java Class>>
mMemoizer = new Memoizer<>
                                                                      MainActivity
      (PrimeCheckers::bruteForceChecker);
                                                            onCreate(Bundle):void
                                                            onRetainNonConfigurationInstance():Object
                                                            ■ initializeViews():void
                                                            setCount(View):void
new Random()
                                                            startOrStopComputations(View);void
                                                            startComputations(int):void
      .longs(count,
                                                            interruptComputations():void
                sMAX VALUE - count,
                                                            odone():void
                                                            println(String):void
                SMAX VALUE)
                                                            onDestroy():void
      .mapToObj(ranNum ->
          new PrimeCallable(ranNum, mMemoizer))
```

```
.forEach(callable ->
    mRetainedState.mExecutorCompService::submit); ...
```

Memoizer caches results when processing a stream of PrimeCallables

```
mMemoizer = new Memoizer<>
                                                                                <<Java Interface>>
                                                                               Function<T.R>
       (PrimeCheckers::bruteForceChecker);
                                                                    apply(T)
                                                                    compose(Function<? super V,? extends T>):Function<V,R>
                                                                    andThen(Function<? super R,? extends V>):Function<T,V>
          This memoizer caches prime # results
                                                                    sidentity():Function<T,T>
                                                                               -mFunction / 0.5
new Random()
       .longs(count,
                                                                                 <<Java Class>>

⊕ Memoizer<K.V>

                  sMAX VALUE - count,

√TAG: String

                  SMAX VALUE)
                                                                          FmCache: ConcurrentMap<K,V>
                                                                          FmFunction: Function<K.V>
       .mapToObj(ranNum ->
                                                                          Memoizer(Function<K,V>,Map<K,V>)
                                                                          apply(K)
           new PrimeCallable(ranNum, mMemoizer))
       .forEach(callable ->
```

See PrimeExecutorServiceFutureTask/app/src/main/java/vandy/mooc/prime/utils/Memoizer.java

mRetainedState.mExecutorCompService::submit); ...

```
mMemoizer = new Memoizer<>
                                                                            <<Java Interface>>
                                                                            Function<T.R>
      (PrimeCheckers::bruteForceChecker);
                                                                 apply(T)
                                                                 compose(Function<? super V,? extends T>):Function<V,R>
                                                                 andThen(Function<? super R,? extends V>):Function<T,V>
                                                                 Sidentity():Function<T,T>
It's easy to change the prime # checker from this...
                                                                           -mFunction / 0.5
new Random()
      .longs(count,
                                                                             <<Java Class>>

⊕ Memoizer<K.V>

                 sMAX VALUE - count,

√TAG: String

                 SMAX VALUE)
                                                                       FmCache: ConcurrentMap<K,V>
                                                                       FmFunction: Function<K.V>
      .mapToObj(ranNum ->
                                                                       Memoizer(Function<K,V>,Map<K,V>)
                                                                       apply(K)
          new PrimeCallable(ranNum, mMemoizer))
      .forEach(callable ->
                    mRetainedState.mExecutorCompService::submit);
```

```
mMemoizer = new Memoizer<>
                                                                             <<Java Interface>>
                                                                             Function<T.R>
      (PrimeCheckers::efficientChecker);
                                                                  apply(T)
                                                                  compose(Function<? super V,? extends T>):Function<V,R>
                                                                  andThen(Function<? super R,? extends V>):Function<T,V>
                                                                  Sidentity():Function<T,T>
                                   ..to this...
                                                                            -mFunction / 0.5
new Random()
      .longs(count,
                                                                               <<Java Class>>

⊕ Memoizer<K.V>

                 sMAX VALUE - count,

√TAG: String

                 SMAX VALUE)
                                                                        FmCache: ConcurrentMap<K,V>
                                                                        FmFunction: Function<K.V>
      .mapToObj(ranNum ->
                                                                        Memoizer(Function<K,V>,Map<K,V>)
                                                                        apply(K)
           new PrimeCallable(ranNum, mMemoizer))
      .forEach(callable ->
                    mRetainedState.mExecutorCompService::submit);
```

```
mMemoizer = new Memoizer<>
    (PrimeCheckers::bruteForceChecker);
                                Generates "count" random #'s between
                                 SMAX_VALUE - count & SMAX_VALUE
new Random()
    .longs(count,
           sMAX VALUE - count,
           SMAX VALUE)
    .mapToObj(ranNum ->
       new PrimeCallable(ranNum, mMemoizer))
    .forEach(callable ->
             mRetainedState.mExecutorCompService::submit); ...
```

```
mMemoizer = new Memoizer<>
    (PrimeCheckers::bruteForceChecker);
                            Transforms random
                           #'s into PrimeCallables
new Random()
    .longs(count,
            sMAX VALUE - count,
            SMAX VALUE)
    .mapToObj(ranNum ->
       new PrimeCallable(ranNum, mMemoizer))
    .forEach(callable ->
              mRetainedState.mExecutorCompService::submit); ...
```

```
mMemoizer = new Memoizer<>
                                                                                                                                                                                                                                                                                                                                                 <<Java Class>>
                              (PrimeCheckers::bruteForceChecker);
                                                                                                                                                                                                                                                                                                                                              PrimeCallable

♣ PrimeCallable(lon | Function | Long, Long | Function | Func
                                                                                                                                                                                                                                                                                                       call():PrimeResult
                                                                                                                                                                                    This memoizer can be used
new Random()
                                                                                                                                                                      wherever a Function is expected
                             .longs(count,
                                                                            sMAX VALUE - count,
                                                                                                                                                                                                                                                                                                                                               <<Java Class>>
                                                                                                                                                                                                                                                                                                                                               PrimeResult
                                                                            SMAX VALUE)
                                                                                                                                                                                                                                                                                                                                   ▲ PrimeResult(long,long)
                             .mapToObj(ranNum ->
                                                new PrimeCallable(ranNum, mMemoizer))
                              .forEach(callable ->
                                                                                         mRetainedState.mExecutorCompService::submit); ...
```

Memoizer caches results when processing a stream of PrimeCallables

(PrimeCheckers::bruteForceChecker);

mMemoizer = new Memoizer<>

```
new Random()
       .longs(count,
                                               Submit a value-returning task for
               sMAX VALUE - count,
                                               execution for each prime callable
               SMAX VALUE)
       .mapToObj(ranNum ->
          new PrimeCallable(ranNum, mMemoizer))
       .forEach(callable ->
                 mRetainedState.mExecutorCompService::submit); ...
See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorCompletionService.html#submit
```

Memoizer caches results when processing a stream of PrimeCallables

new PrimeCallable(ranNum, mMemoizer))

```
mMemoizer = new Memoizer<>
    (PrimeCheckers::bruteForceChecker);
```

.forEach(callable ->

mRetainedState.mExecutorCompService::submit); ...

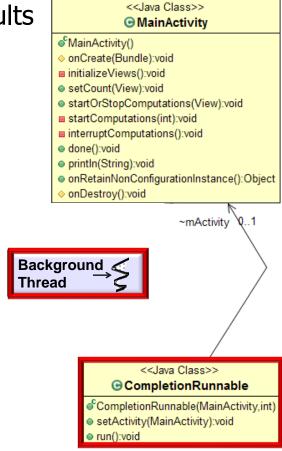
 MainActivity creates a thread to wait for all future results in the background so the UI thread doesn't block

```
mRetainedState.mCompletionRunnable =
  new CompletionRunnable(this, count);
```

CompletionRunnable is stored in a field so it can be updated during a runtime configuration change

```
mRetainedState.mThread = new Thread
  (mRetainedState.mCompletionRunnable);
```

```
mRetainedState.mThread.start();
```



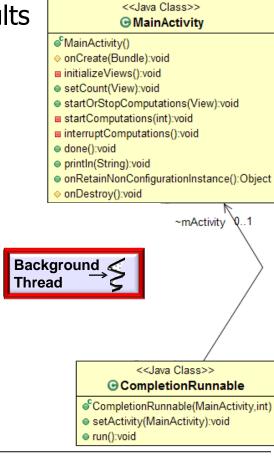
 MainActivity creates a thread to wait for all future results in the background so the UI thread doesn't block

```
mRetainedState.mCompletionRunnable =
  new CompletionRunnable(this, count);
```

```
A new thread is created/started to execute the CompletionRunnable
```

```
mRetainedState.mThread = new Thread
  (mRetainedState.mCompletionRunnable);
```

```
mRetainedState.mThread.start();
```



<<Java Class>> CompletionRunnable gets results as futures complete • MainActivity class CompletionRunnable implements Runnable { onCreate(Bundle):void ■ initializeViews():void int mCount; setCount(View):void startOrStopComputations(View):void MainActivity mActivity; ... startComputations(int):void ■ interruptComputations():void odone():void println(String):void public void run() { onRetainNonConfigurationInstance():Object for (int i = 0; i < mCount; ++i) { onDestroy():void PrimeResult pr = ... ~mActivity mExecutorCompService Background .take().get(); **Thread** if (pr.mSmallestFactor != 0) ... else ... <<.lava Class>> ⊕ CompletionRunnable CompletionRunnable(MainActivity,int) setActivity(MainActivity):void mActivity.done(); ... o run():void

<<Java Class>> CompletionRunnable gets results as futures complete MainActivity class CompletionRunnable implements Runnable { onCreate(Bundle):void ■ initializeViews():void int mCount; setCount(View):void startOrStopComputations(View):void Iterate thru MainActivity mActivity; ... startComputations(int):void all results ■ interruptComputations():void odone():void public void run() { println(String):void onRetainNonConfigurationInstance():Object for (int i = 0; i < mCount; ++i) { onDestroy():void PrimeResult pr = ... ~mActivity mExecutorCompService Background . .take().get(); **Thread** if (pr.mSmallestFactor != 0) ... else ... <<.lava Class>> ⊕ CompletionRunnable CompletionRunnable(MainActivity,int) setActivity(MainActivity):void mActivity.done(); ... o run():void

<<Java Class>> CompletionRunnable gets results as futures complete • MainActivity class CompletionRunnable implements Runnable { onCreate(Bundle):void ■ initializeViews():void int mCount; setCount(View):void startOrStopComputations(View):void MainActivity mActivity; ... startComputations(int):void ■ interruptComputations():void odone():void public void run() { println(String):void onRetainNonConfigurationInstance():Object for (int i = 0; i < mCount; ++i) { onDestroy():void PrimeResult pr = ... ~mActivity **mExecutorCompService** Background .take().get(); **Thread** (pr.mSmallestFactor\!= 0) ... else <<.lava Class>> ⊕ CompletionRunnable get() doesn't block, though take() may block CompletionRunnable(MainActivity,int) if completed futures aren't yet available setActivity(MainActivity):void mActivity. o run():void

<<Java Class>> CompletionRunnable gets results as futures complete MainActivity class CompletionRunnable implements Runnable { onCreate(Bundle):void ■ initializeViews():void int mCount; setCount(View):void startOrStopComputations(View):void MainActivity mActivity; ... startComputations(int):void ■ interruptComputations():void odone():void public void run() { println(String):void onRetainNonConfigurationInstance():Object for (int i = 0; i < mCount; ++i) { onDestroy():void PrimeResult pr = ... ~mActivity **mExecutorCompService** Background . .take().get(); **Thread** (pr.mSmallestFactor != 0) ... else ... <<.lava Class>> ⊕ CompletionRunnable Process & output results CompletionRunnable(MainActivity,int) setActivity(MainActivity):void mActivity.done(); o run():void

<<Java Class>>

MainActivity

o run():void

 RetainedState maintains key concurrency state across runtime configuration changes

```
MainActivity()
                                                                                         onCreate(Bundle):void
                                                                                         ■ initializeViews():void
class RetainedState {
                                                                                         setCount(View):void
                                                                                         startOrStopComputations(View):void
   ExecutorCompletionService
                                                                                         startComputations(int):void
                                                                                         ■ interruptComputations():void
       mExecutorCompService;
                                                                                         odone():void
                                                                                         println(String):void
                                                                                         onRetainNonConfigurationInstance():Object
                                                                                         onDestroy():void
   ExecutorService mExecutorService;
                                                                                                         ~mActivity
                                                                                  -mRetainedState 10..1
   CompletionRunnable mCompletionRunnable;
                                                                                      <<Java Class>>
                                                                                     Retained State
                                                                                     RetainedState()
   Thread mThread;
                                                                                   ~mCompletionRunnable, 0..1
                                                                                                    <<.lava Class>>
   Memoizer<Long, Long> mMemoizer;
                                                                                                G CompletionRunnable
                                                                                             CompletionRunnable(MainActivity,int)
                                                                                             setActivity(MainActivity):void
```

See android.jlelse.eu/handling-orientation-changes-in-android-7072958c442a

RetainedState maintains key concurrency state across runtime configuration changes

return mRetainedState; }

```
void onCreate(...) {
  mRetainedState = (RetainedState)
     getLastNonConfigurationInstance();
                                                             odone():void
  if (mRetainedState != null) {
     ... // update configurations
                                                         -mRetainedState 10...1
                                                           <<Java Class>>
       Android's activity framework dispatches these
                                                           Retained State
                                                           RetainedState()
        hook methods to save & restore state when
           runtime configuration changes occur
Object onRetainNonConfigurationInstance()
```

MainActivity() onCreate(Bundle):void ■ initializeViews():void setCount(View):void startOrStopComputations(View):void startComputations(int):void ■ interruptComputations():void println(String):void onRetainNonConfigurationInstance():Object onDestroy():void ~mActivity ~mCompletionRunnable, 0..1 <<.lava Class>> G CompletionRunnable CompletionRunnable(MainActivity,int) setActivity(MainActivity):void o run():void

<<Java Class>>

• MainActivity

See android.jlelse.eu/handling-orientation-changes-in-android-7072958c442a

# End of Java FutureTask: Applying Memoizer to the PrimeChecker App