Java ExecutorCompletionService: Application to PrimeChecker App

Douglas C. Schmidt

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

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

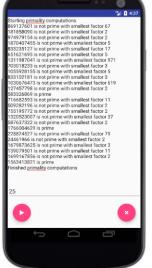
Institute for Software Integrated Systems

Vanderbilt University Nashville, Tennessee, USA

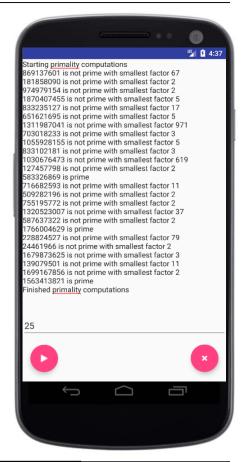


Learning Objectives in this Part of the Lesson

- Understand how the Java CompletionService interface defines a framework for handling the completion of asynchronous tasks
- Know how to instantiate the Java ExecutorCompletionService
- Recognize key methods in the Java CompletionService interface
- Visualize the ExecutorCompletionService in action
- Be aware of how the Java ExecutorCompletionService implements the CompletionService interface
- Know how to apply the Java ConcurrentHashMap class to design a "memoizer"
- Master how to implement the Memoizer class with Java ConcurrentHashMap
- See how Java ExecutorCompletionService & Memoizer are integrated into the "PrimeChecker" app

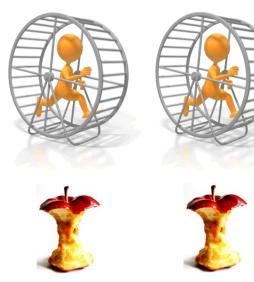


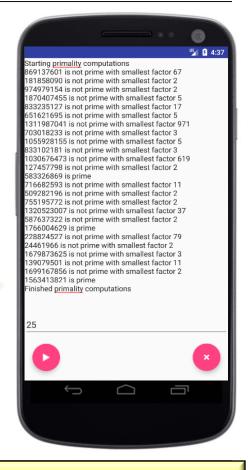
 This app shows how Java's ExecutorCompletionService framework & the Java 8-based Memoizer can be used to check if N random #'s are prime



- This app shows how Java's ExecutorCompletionService framework & the Java 8-based Memoizer can be used to check if N random #'s are prime
 - This app is "embarrassingly parallel" & compute-bound







<<Java Class>> MainActivity checks primality of "count" random #'s via an MainActivity ExecutorService w/a thread pool & the PrimeCallable class MainActivity() onCreate(Bundle):void ■ initializeViews():void setCount(View):void startOrStopComputations(View):void <<Java Class>> startComputations(int):void PrimeCallable Cached (VariableinterruptComputations():void PrimeCallable(long,Function<Long,Long>) odone():void sized) Thread Pool call():PrimeResult println(String):void A pool of worker threa onRetainNonConfigurationInstance():Object onDestroy():void ~mActivity mExecutor = Executors-mRetainedState 10 .newCachedThreadPool(); <<Java Class>> <<Java Class>> Retained State **©** PrimeResult RetainedState() ▲ PrimeResult(long,long) ~mCompletionRunnable, 0..1 The executor service uses a cached <<.lava Class>> (variable-sized) pool of threads • CompletionRunnable

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

CompletionRunnable(MainActivity,int)
 setActivity(MainActivity):void

o run():void

```
class PrimeCallable implements Callable<PrimeResult> {
  mFunction<Long, Long> mPrimeChecker;
                                                           <<Java Class>>
                                                          PrimeCallable
                                                   PrimeCallable(long,Function<Long,Long>)
                                                   call():PrimeResult
  PrimeCallable (Long primeCandidate,
                   Function<Long, Long> pc)
  { mPrimeChecker = pc; }
                                                          <<Java Class>>
                                                          @PrimeResult
  PrimeResult call()
                                                        PrimeResult(long,long)
    return new PrimeResult
       (mPrimeCandidate, mPrimeChecker.apply(mPrimeCandidate));
```

```
class PrimeCallable implements Callable<PrimeResult> {
  mFunction<Long, Long> mPrimeChecker;
                                                           <<Java Class>>
                                                          PrimeCallable
        Implements Callable to run in a pool thread
                                                   PrimeCallable(long,Function<Long,Long>)
                                                   call():PrimeResult
  PrimeCallable (Long primeCandidate,
                   Function<Long, Long> pc)
   mPrimeChecker = pc; }
                                                          <<Java Class>>
                                                          PrimeResult
  PrimeResult call()
                                                        PrimeResult(long,long)
    return new PrimeResult
       (mPrimeCandidate, mPrimeChecker.apply(mPrimeCandidate));
```

 PrimeCallable defines a two-way means of determining whether a # is prime by calling a function that returns 0 if it's prime or smallest factor if it's not

```
class PrimeCallable implements Callable<PrimeResult> {
   mFunction<Long, Long> mPrimeChecker;
                                                                 <<Java Class>>
                                                                 PrimeCallable
                                                         PrimeCallable(long,Function<Long,Long>)
The function computing primality is parameterized
                                                         call():PrimeResult
```

```
PrimeCallable (Long primeCandidate,
                Function<Long, Long> pc)
{ mPrimeChecker = pc; }
                                                      <<Java Class>>
                                                      PrimeResult
PrimeResult call() {
                                                    PrimeResult(long,long)
  return new PrimeResult
```

(mPrimeCandidate, mPrimeChecker.apply(mPrimeCandidate));

This Function param is a new feature added since the earlier PrimeCheck example

```
class PrimeCallable implements Callable<PrimeResult> {
  mFunction<Long, Long> mPrimeChecker;
                                                            <<Java Class>>
                                                           PrimeCallable
                                                    PrimeCallable(long,Function<Long,Long>)
                                                    call():PrimeResult
  PrimeCallable (Long primeCandidate,
                   Function<Long, Long> pc)
    mPrimeChecker = pc; }
                                  This hook method is
                                                           <<Java Class>>
                                 called in a pool thread
                                                           PrimeResult
  PrimeResult call()

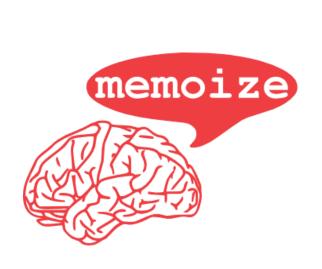
▲ PrimeResult(long,long)

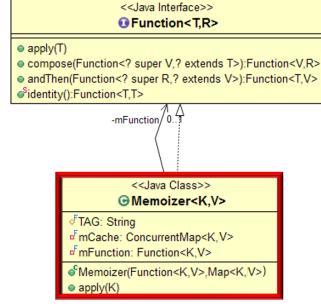
     return new PrimeResult
        (mPrimeCandidate, mPrimeChecker.apply(mPrimeCandidate));
```

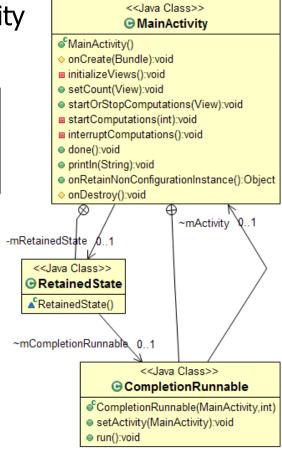
```
class PrimeCallable implements Callable<PrimeResult> {
  mFunction<Long, Long> mPrimeChecker;
                                                           <<Java Class>>
                                                          PrimeCallable
                                                   PrimeCallable(long,Function<Long,Long>)
                                                   call():PrimeResult
  PrimeCallable (Long primeCandidate,
                   Function<Long, Long> pc)
    mPrimeChecker = pc; }
                                This function performs
                                                          <<Java Class>>
                                  the prime # check
  PrimeResult call()
                                                          PrimeResult
                                                        PrimeResult(long,long)
     return new PrimeResult
       (mPrimeCandidate, mPrimeChecker.apply(mPrimeCandidate));
```

```
class PrimeCallable implements Callable<PrimeResult> {
   mFunction<Long, Long> mPrimeChecker;
                                                            <<Java Class>>
                                                           PrimeCallable
                                                    PrimeCallable(long,Function<Long,Long>)
Match prime # candidate with primality check result
                                                    call():PrimeResult
   PrimeCallable (Long primeCandidate,
                    Function<Long, Long> pc)
     mPrimeChecker = pc; }
                                                           <<Java Class>>
                                                           PrimeResult
   PrimeResult call()
                                                          PrimeResult(long,long)
     return new PrimeResult
        (mPrimeCandidate, mPrimeChecker.apply(mPrimeCandidate));
```

 MainActivity creates a Memoizer that optimizes primality checking of "count" random #'s







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

Memoizer caches results when processing a stream of PrimeCallables

```
mMemoizer = new Memoizer<>
                                                                 <<Java Class>>
                                                                 MainActivity
      (PrimeCheckers::bruteForceChecker,
                                                        new ConcurrentHashMap());
                                                        onCreate(Bundle):void
                                                        onRetainNonConfigurationInstance():Object
                                                        ■ initializeViews():void
                                                        setCount(View):void
new Random()
                                                        startOrStonComputations(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)
       new ConcurrentHashMap());
                                                                compose(Function<? super V,? extends T>):Function<V,R>
                                                                andThen(Function<? super R,? extends V>):Function<T,V>
                                                                Sidentity():Function<T,T>
         This memoizer caches prime # results
                                                                          -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);
```

Memoizer caches results when processing a stream of PrimeCallables

.forEach(callable ->

```
mMemoizer = new Memoizer<>
                                                                               <<Java Interface>>
                                                                               Function<T.R>
       (PrimeCheckers::bruteForceChecker,
                                                                    apply(T)
        new ConcurrentHashMap());
                                                                    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 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))
```

mRetainedState.mExecutorCompService::submit); ...

See blog.indrek.io/articles/java-8-behavior-parameterization

Memoizer caches results when processing a stream of PrimeCallables

.forEach(callable ->

```
mMemoizer = new Memoizer<>
                                                                                 <<Java Interface>>
                                                                                 Function<T.R>
       (PrimeCheckers::efficientChecker,
                                                                     apply(T)
        new ConcurrentHashMap());
                                                                     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))
```

See blog.indrek.io/articles/java-8-behavior-parameterization

mRetainedState.mExecutorCompService::submit); ...

Memoizer caches results when processing a stream of PrimeCallables

```
mMemoizer = new Memoizer<>
                                                                               <<Java Interface>>
                                                                               Function<T.R>
       (PrimeCheckers::efficientChecker,
                                                                    apply(T)
        new ConcurrentHashMap());
                                                                    compose(Function<? super V,? extends T>):Function<V,R>
                                                                    andThen(Function<? super R,? extends V>):Function<T,V>
                                                                    sidentity():Function<T,T>
                                                                              -mFunction/ 0.1
                      Ensure efficient thread-safe map operations
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))
```

mRetainedState.mExecutorCompService::submit); ...

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

.forEach(callable ->

Memoizer caches results when processing a stream of PrimeCallables

```
mMemoizer = new Memoizer<>
    (PrimeCheckers::efficientChecker,
     new ConcurrentHashMap());
                                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); ...
```

Memoizer caches results when processing a stream of PrimeCallables

```
mMemoizer = new Memoizer<>
    (PrimeCheckers::efficientChecker,
     new ConcurrentHashMap());
                            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); ...

Memoizer caches results when processing a stream of PrimeCallables

```
mMemoizer = new Memoizer<>
                                                             <<Java Class>>
     (PrimeCheckers::efficientChecker,
                                                             PrimeCallable

♣ PrimeCallable(lon , Function<Long,Long>)

      new ConcurrentHashMap());
                                                      call():PrimeResult
                              A Memoizer object 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::efficientChecker,

new ConcurrentHashMap());

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

```
mMemoizer = new Memoizer<>
    (PrimeCheckers::efficientChecker,
     new ConcurrentHashMap());
new Random()
    .longs(count,
                                       There's no need for a list of futures
            sMAX VALUE - count,
                                     due to the ExecutorCompletionService
            SMAX VALUE)
    .mapToObj(ranNum ->
       new PrimeCallable(ranNum, mMemoizer))
    .forEach(callable ->
              mRetainedState.mExecutorCompService::submit); ...
```

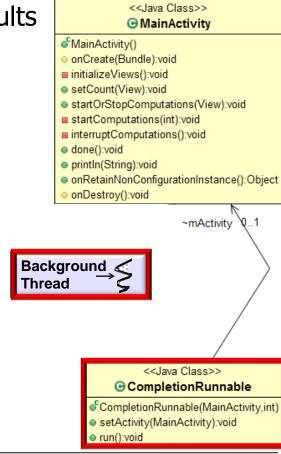
 MainActivity creates a thread to wait for all future results in the background so the UI thread doesn't block

```
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();
```



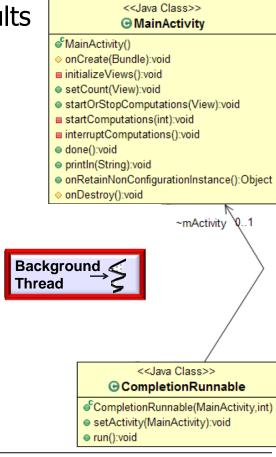
 MainActivity creates a thread to wait for all future results in the background so the UI thread doesn't block

```
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();
```



CompletionRunnable gets results as futures complete

int mCount;

class CompletionRunnable implements Runnable {

<<Java Class>>

• MainActivity

onCreate(Bundle):void
 initializeViews():void

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** if (pr.mSmallestFactor != 0) ... else ... <<.lava Class>> G CompletionRunnable CompletionRunnable(MainActivity,int) setActivity(MainActivity):void mActivity.done(); ... run():void See src/main/java/vandy/mooc/prime/activities/PrimeCallable.java

CompletionRunnable gets results as futures complete

<<Java Class>>

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** if (pr.mSmallestFactor != 0) ... else ... <<.lava Class>> G CompletionRunnable CompletionRunnable(MainActivity,int) setActivity(MainActivity):void mActivity.done(); ... o run():void See docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html

<<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 no completed futures are 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** if (pr.mSmallestFactor != 0) ... else ... <<.lava Class>> ⊕ CompletionRunnable Process & output results CompletionRunnable(MainActivity.int) setActivity(MainActivity):void mActivity.done(); 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;
                                                                               -mRetainedState 10..1
   CompletionRunnable mCompletionRunnable;
                                                                                  <<Java Class>>
                                                                                 Retained State
                                                                                  RetainedState()
   Thread mThread;
                                                                                ~mCompletionRunnable, 0..1
                                                                                                <<.lava Class>>
   Memoizer<Long, Long> mMemoizer;

⊕ CompletionRunnable
```

CompletionRunnable(MainActivity,int) setActivity(MainActivity):void o run():void See android.jlelse.eu/handling-orientation-changes-in-android-7072958c442a

<<Java Class>>

• MainActivity

~mActivity

RetainedState maintains key concurrency state across runtime configuration changes

```
void onCreate(...) {
  mRetainedState = (RetainedState)
   getLastNonConfigurationInstance();
  if (mRetainedState != null) {
    ... // update configurations
      Android's activity framework dispatches these
       hook methods to save & restore state when
          runtime configuration changes occur
Object onRetainNonConfigurationInstance()
```

return mRetainedState; }

• MainActivity onCreate(Bundle):void ■ initializeViews():void setCount(View):void startOrStopComputations(View):void startComputations(int):void ■ interruptComputations():void odone():void println(String):void onRetainNonConfigurationInstance():Object onDestroy():void ~mActivity -mRetainedState 10...1 <<Java Class>> Retained State RetainedState() ~mCompletionRunnable, 0..1 <<.lava Class>> **⊕** CompletionRunnable CompletionRunnable(MainActivity,int) setActivity(MainActivity):void o run():void

<<Java Class>>

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

End of Java Executor CompletionService: Application to PrimeChecker App