Applying TimedMemoizerEx to the PrimeChecker App

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

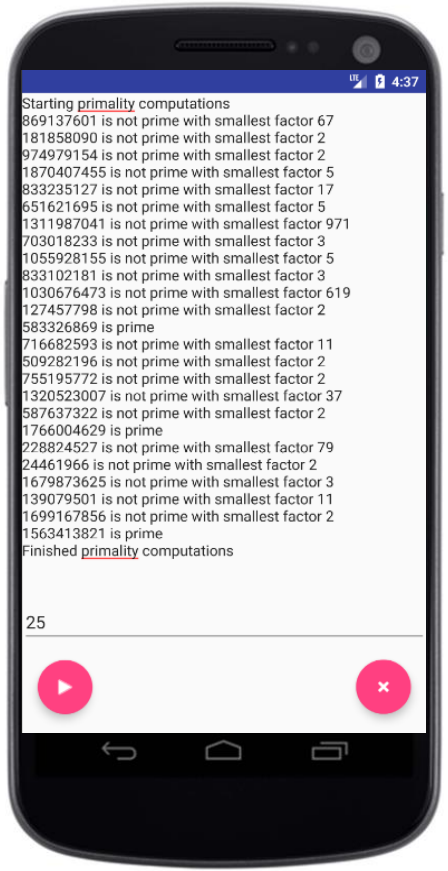
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

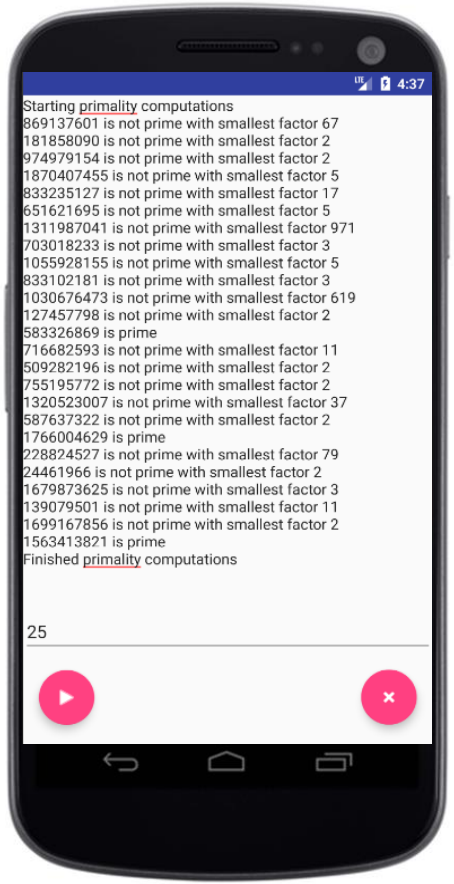
- See how the TimedMemoizerEx is integrated into the "PrimeChecker" app



Applying TimedMemoizerEx to Check for Prime #'s

Applying TimedMemoizerEx to Check for Prime #'s

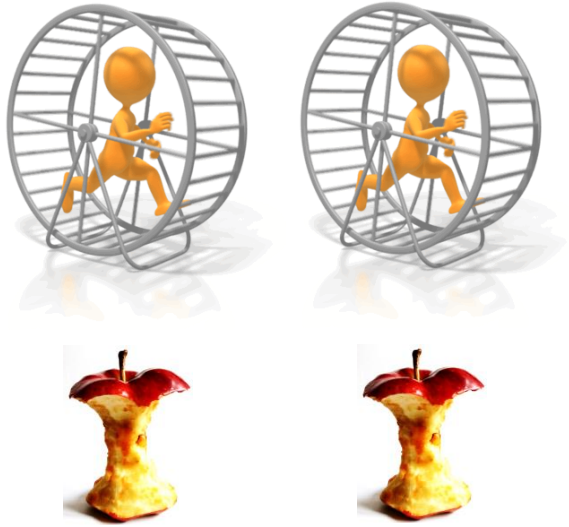
- This app shows how the Java ExecutorCompletionService framework & TimedMemoizerEx can check if *N* random #'s are prime

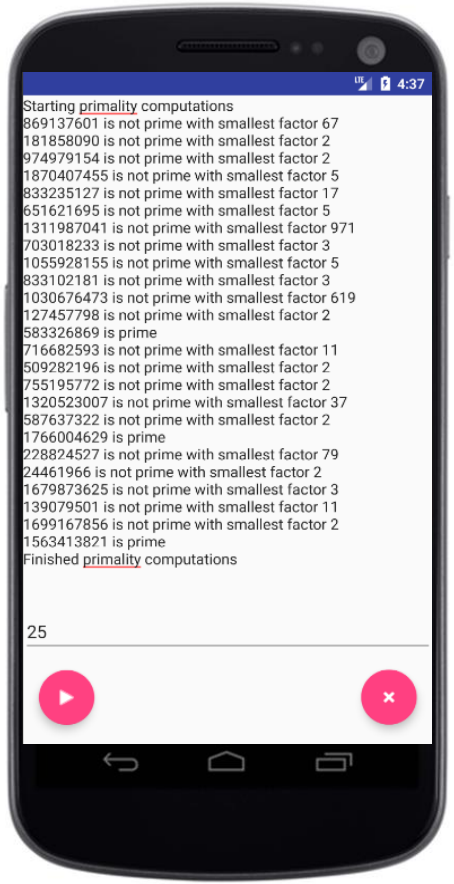


See github.com/douglascraigschmidt/POSA/tree/master/ex/M4/Primes/PrimeScheduledExecutorService

Applying TimedMemoizerEx to Check for Prime #'s

- This app shows how the Java ExecutorCompletionService framework & TimedMemoizerEx can check if *N* random #'s are prime
  - This app is "embarrassingly parallel" & compute-bound

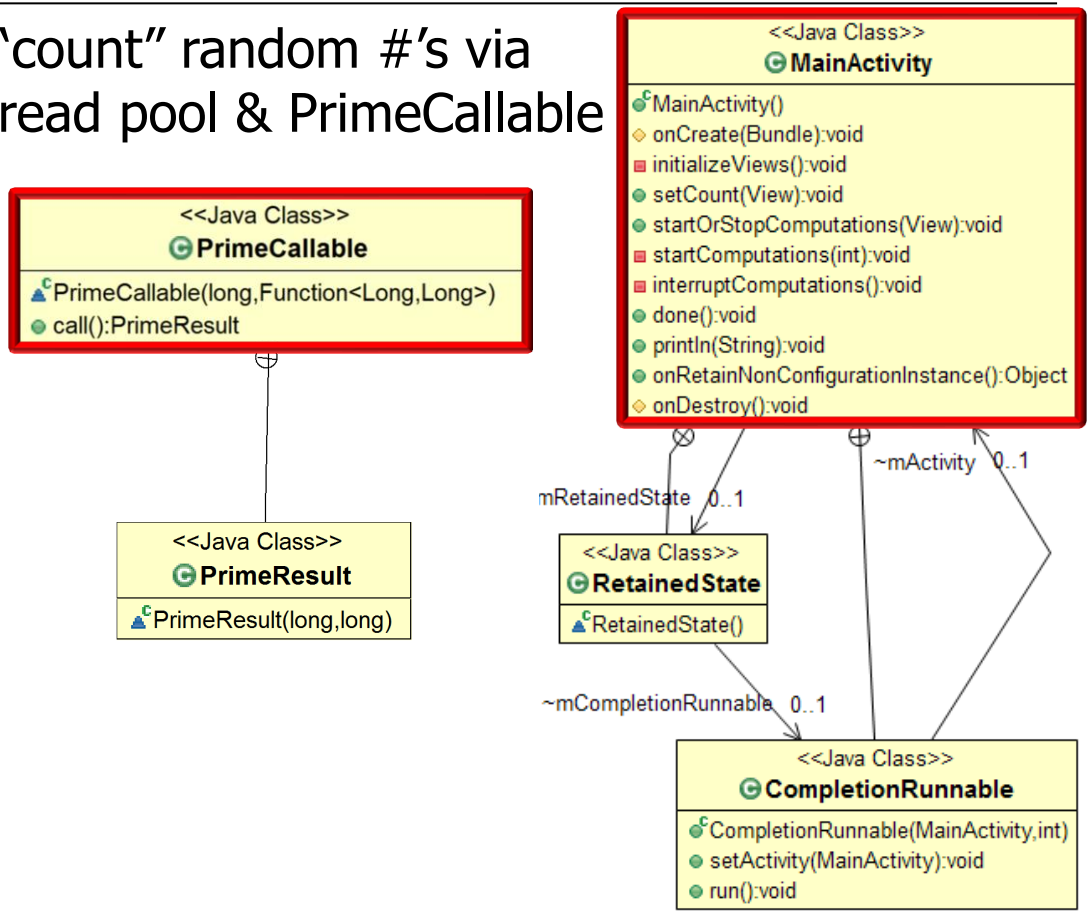




See en.wikipedia.org/wiki/Embarrassingly_parallel & en.wikipedia.org/wiki/CPU-bound

Applying TimedMemoizerEx to Check for Prime #'s

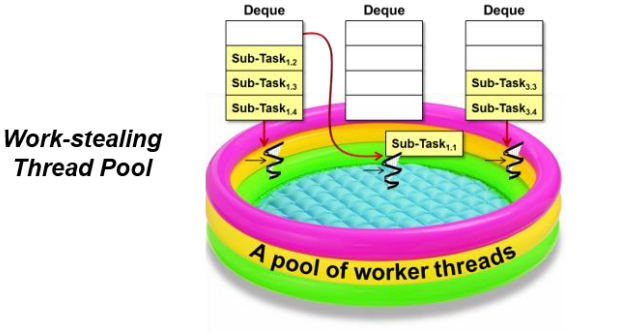
- MainActivity checks primality of "count" random #'s via ExecutorCompletionService w/thread pool & PrimeCallable

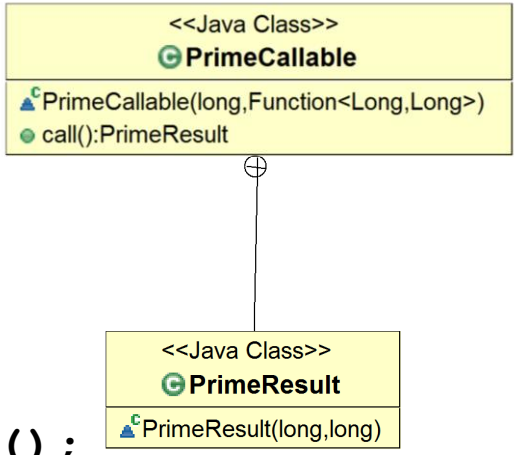


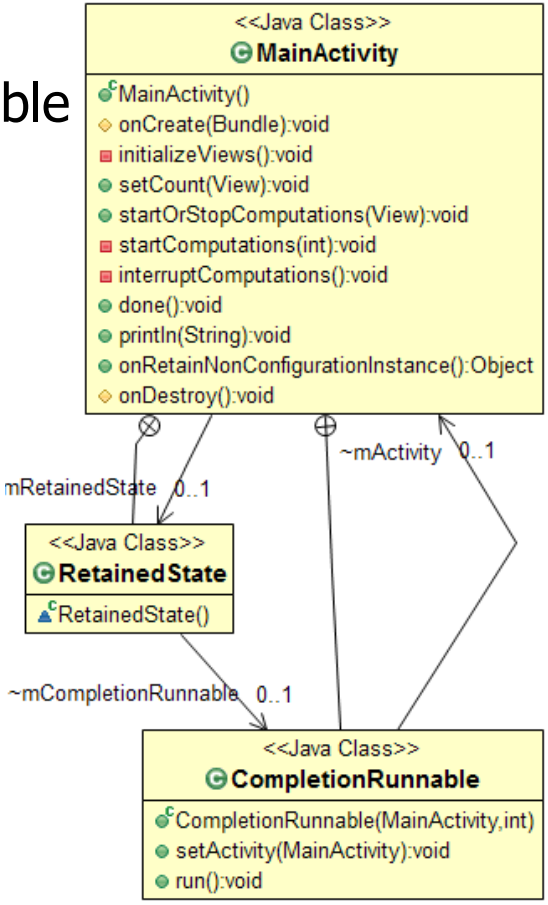
See src/main/java/vandy/mooc/prime/activities/MainActivity.java

Applying TimedMemoizerEx to Check for Prime #'s

- MainActivity checks primality of "count" random #'s via ExecutorCompletionService w/thread pool & PrimeCallable







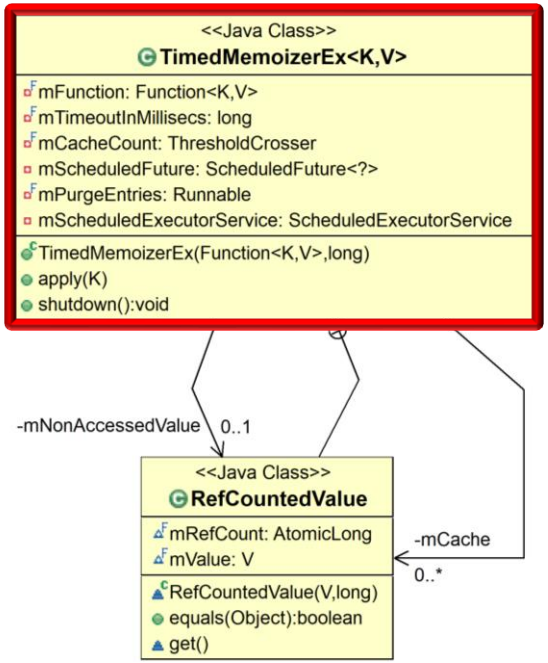

```
mExecutorService = Executors
   .newWorkStealingThreadPool();
```

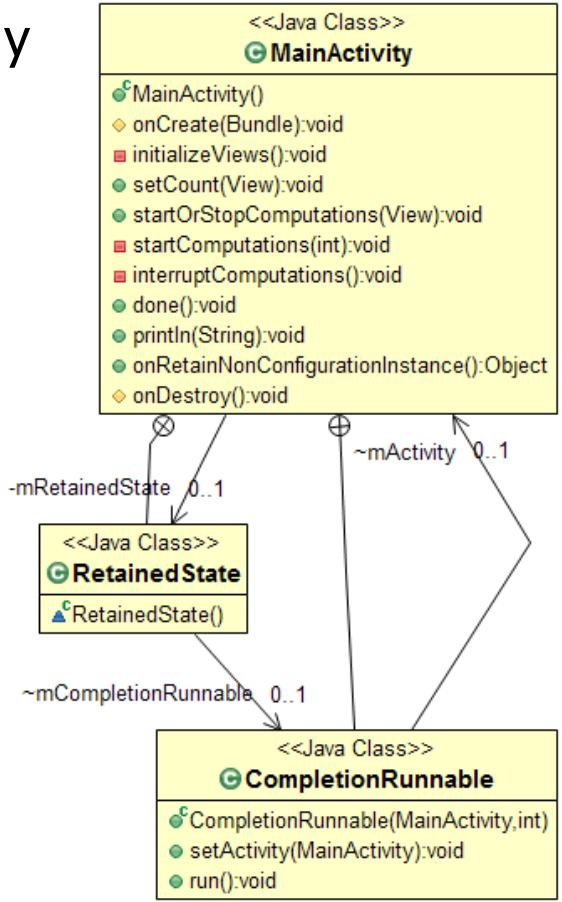
The executor service uses a "work-stealing" fork-join thread pool that's tuned to the # of processor cores

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

Applying TimedMemoizerEx to Check for Prime #'s

- MainActivity also uses the TimedMemoizerEx to scalably optimize primality checking of the random #'s





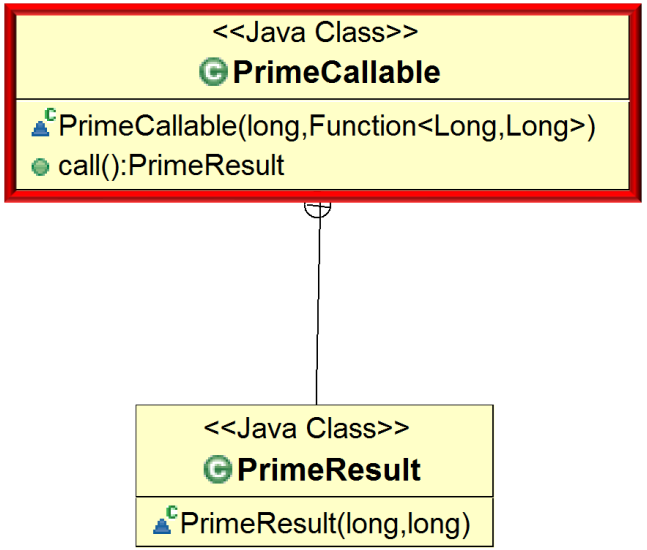
See src/main/java/vandy/mooc/prime/utils/TimedMemoizerEx.java

Applying TimedMemoizerEx to Check for Prime #'s

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

```
class PrimeCallable
      implements Callable<PrimeResult> {
  long mPrimeCandidate;
  mFunction<Long, Long> mPrimeChecker;

  PrimeCallable(Long primeCandidate,
                Function<Long, Long>
                   primeChecker) {
    mPrimeCandidate = primeCandidate;
    mPrimeChecker = primeChecker;
  }
```



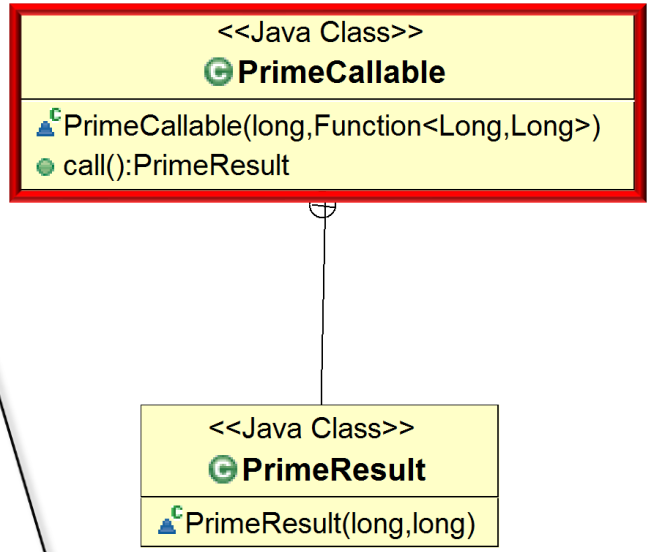
See src/main/java/vandy/mooc/prime/activities/PrimeCallable.java

Applying TimedMemoizerEx to Check for Prime #'s

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

```
class PrimeCallable
      implements Callable<PrimeResult> {
  long mPrimeCandidate;
  mFunction<Long, Long> mPrimeChecker;

  PrimeCallable(Long primeCandidate,
                Function<Long, Long>
                   primeChecker) {
    mPrimeCandidate = primeCandidate;
    mPrimeChecker = primeChecker;
  }
```



PrimeCallable implements the Callable interface so it can be submitted to the Java ExecutorCompletionService framework

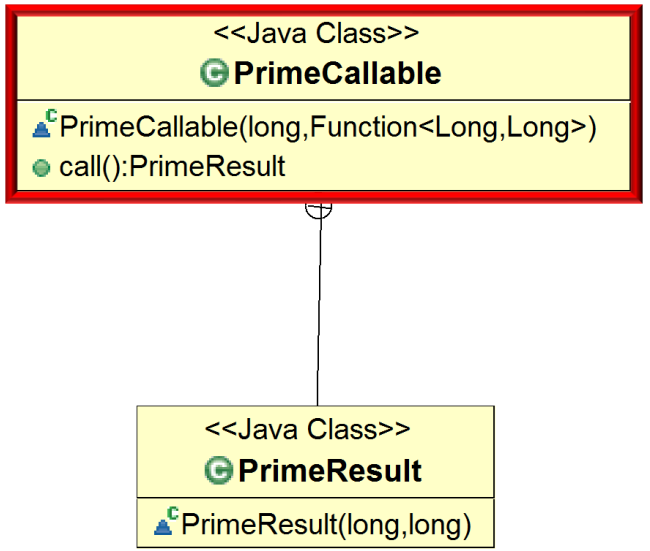
See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html

Applying TimedMemoizerEx to Check for Prime #'s

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

```
class PrimeCallable
      implements Callable<PrimeResult> {
  long mPrimeCandidate;
  mFunction<Long, Long> mPrimeChecker;

  PrimeCallable(Long primeCandidate,
                Function<Long, Long>
                   primeChecker) {
    mPrimeCandidate = primeCandidate;
    mPrimeChecker = primeChecker;
  }
```



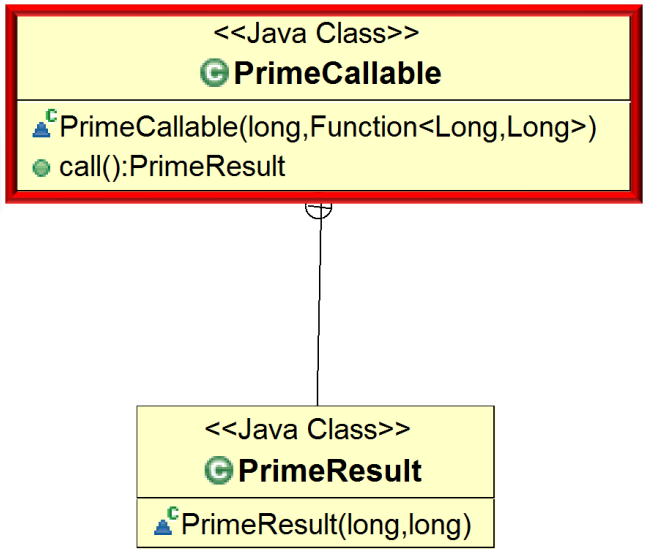
The function that checks primes is passed as a param & stored in a field

Applying TimedMemoizerEx to Check for Prime #'s

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

```
class PrimeCallable
      implements Callable<PrimeResult> {
  ...
  PrimeResult call() {
    return new PrimeResult
      (mPrimeCandidate,
       mPrimeChecker
         .apply(mPrimeCandidate));
  }
  ...
```

The call() hook method runs in a pool thread

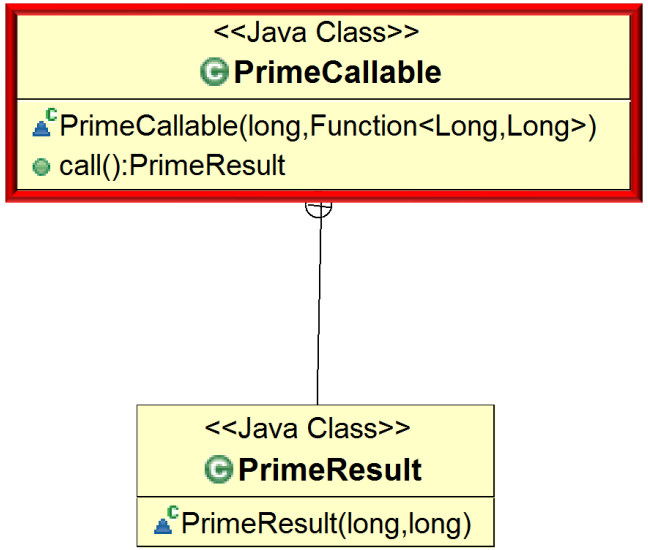


Applying TimedMemoizerEx to Check for Prime #'s

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

```
class PrimeCallable
      implements Callable<PrimeResult> {
  ...

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



apply() returns 0 if the # is prime or smallest factor if it's not

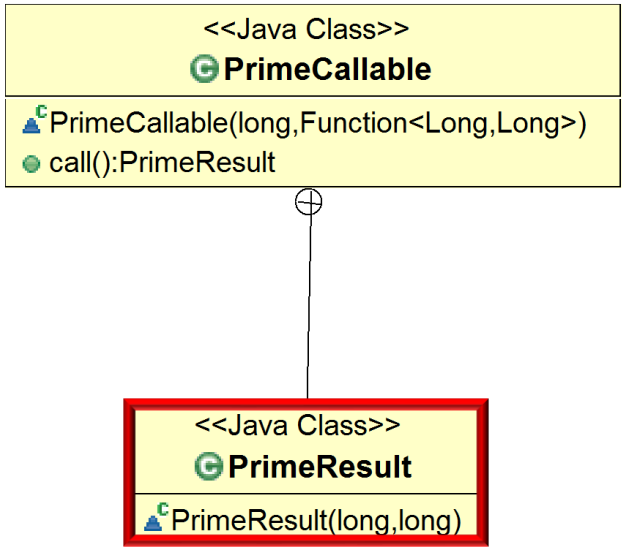
The apply() method call can be transparently optimized via TimedMemoizerEx

Applying TimedMemoizerEx to Check for Prime #'s

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

```
class PrimeCallable
      implements Callable<PrimeResult> {
  ...

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



The PrimeResult tuple matches the prime # candidate with result of checking for primality

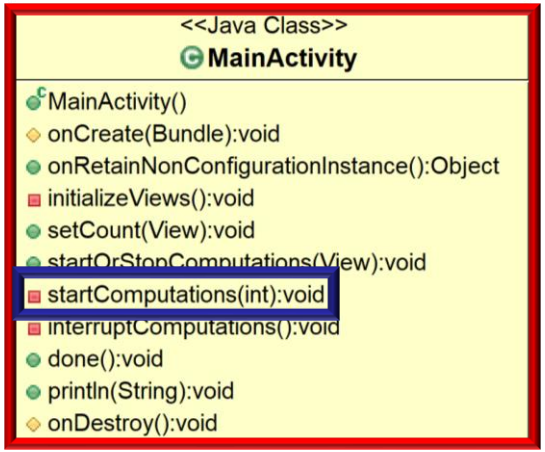
Applying TimedMemoizerEx to Check for Prime #'s

- TimedMemoizerEx caches results when processing a stream of PrimeCallables

```
mTimedMemoizer = new TimedMemoizerEx<>
     (PrimeCheckers::bruteForceChecker,
      count * 500);


new Random()
     .longs(count,
            sMAX_VALUE - count,
            sMAX_VALUE)
     .mapToObj(ranNum ->
        new PrimeCallable(ranNum, mTimedMemoizer))

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



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

Applying TimedMemoizerEx to Check for Prime #'s

- TimedMemoizerEx caches results when processing a stream of PrimeCallables

```
mTimedMemoizer = new TimedMemoizerEx<>
     (PrimeCheckers::bruteForceChecker,
      count * 500);


new Random()
     .longs(count,
            sMAX_VALUE - count,
            sMAX_VALUE)
     .mapToObj(ranNum ->
        new PrimeCallable(ranNum, mTimedMemoizer))

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

*This memoizer caches prime # results & automatically times out stale cache entries after count * 0.5 secs*

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

Applying TimedMemoizerEx to Check for Prime #'s

- TimedMemoizerEx caches results when processing a stream of PrimeCallables

```
mTimedMemoizer = new TimedMemoizerEx<>
     (PrimeCheckers::bruteForceChecker,
      count * 500);


new Random()
     .longs(count,
            sMAX_VALUE - count,
            sMAX_VALUE)
     .mapToObj(ranNum ->
        new PrimeCallable(ranNum, mTimedMemoizer))

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

Generates "count" random #'s between sMAX_VALUE – count & sMAX_VALUE

Applying TimedMemoizerEx to Check for Prime #'s

- TimedMemoizerEx caches results when processing a stream of PrimeCallables

```
mTimedMemoizer = new TimedMemoizerEx<>
     (PrimeCheckers::bruteForceChecker,
      count * 500);


new Random()
     .longs(count,
            sMAX_VALUE - count,
            sMAX_VALUE)
     .mapToObj(ranNum ->
        new PrimeCallable(ranNum, mTimedMemoizer))

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

Transforms random #'s into PrimeCallables

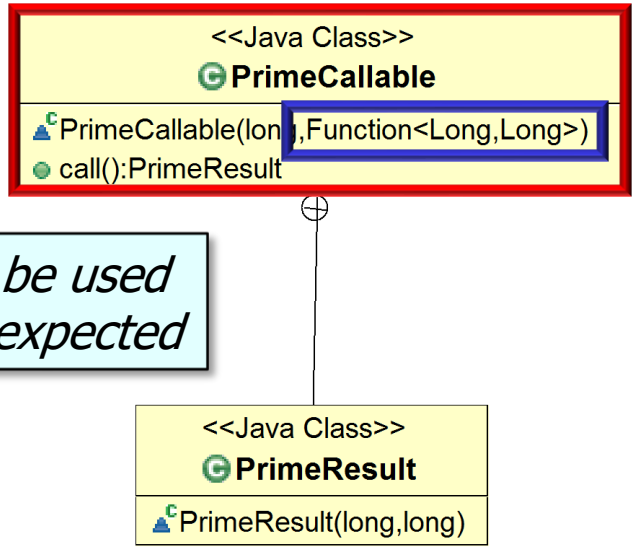
Applying TimedMemoizerEx to Check for Prime #'s

- TimedMemoizerEx caches results when processing a stream of PrimeCallables

```
mTimedMemoizer = new TimedMemoizerEx<>
     (PrimeCheckers::bruteForceChecker,
      count * 500);


new Random()
     .longs(count,
            sMAX_VALUE - count,
            sMAX_VALUE)
     .mapToObj(ranNum ->
        new PrimeCallable(ranNum, mTimedMemoizer))

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



Applying TimedMemoizerEx to Check for Prime #'s

- TimedMemoizerEx caches results when processing a stream of PrimeCallables

```
mTimedMemoizer = new TimedMemoizerEx<>
     (PrimeCheckers::bruteForceChecker,
      count * 500);


new Random()
     .longs(count,
            sMAX_VALUE - count,
            sMAX_VALUE)
     .mapToObj(ranNum ->
        new PrimeCallable(ranNum, mTimedMemoizer))

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

Submit a value-returning task for execution for each prime callable

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorCompletionService.html#submit

Applying TimedMemoizerEx to Check for Prime #'s

- TimedMemoizerEx caches results when processing a stream of PrimeCallables

```
mTimedMemoizer = new TimedMemoizerEx<>
     (PrimeCheckers::bruteForceChecker,
      count * 500);


new Random()
     .longs(count,
            sMAX_VALUE - count,
            sMAX_VALUE)
     .mapToObj(ranNum ->
        new PrimeCallable(ranNum, mTimedMemoizer))

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

There's no need for a list of futures due to the ExecutorCompletionService

Applying TimedMemoizerEx to Check for Prime #'s

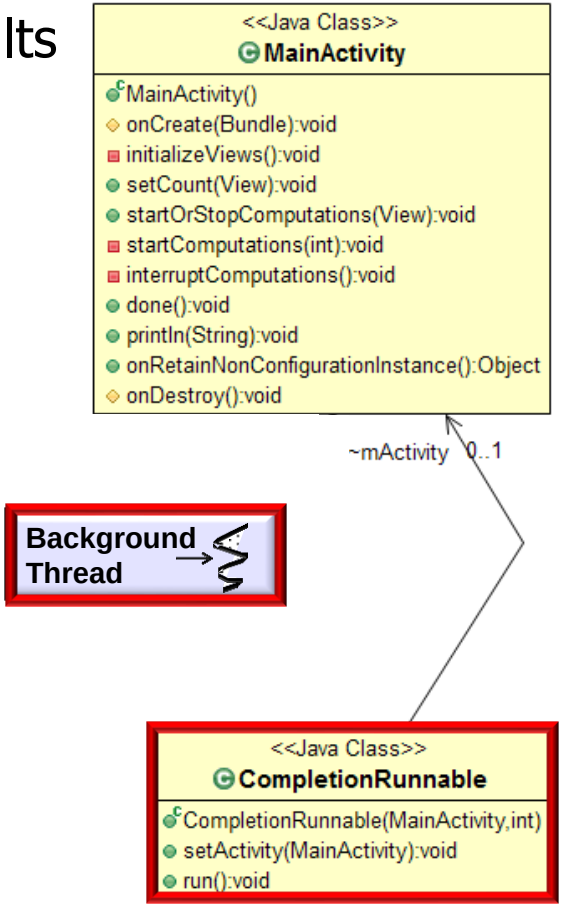
- MainActivity creates a thread to wait for all future results in the background so the UI thread doesn't block

```
...
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();
```



Applying TimedMemoizerEx to Check for Prime #'s

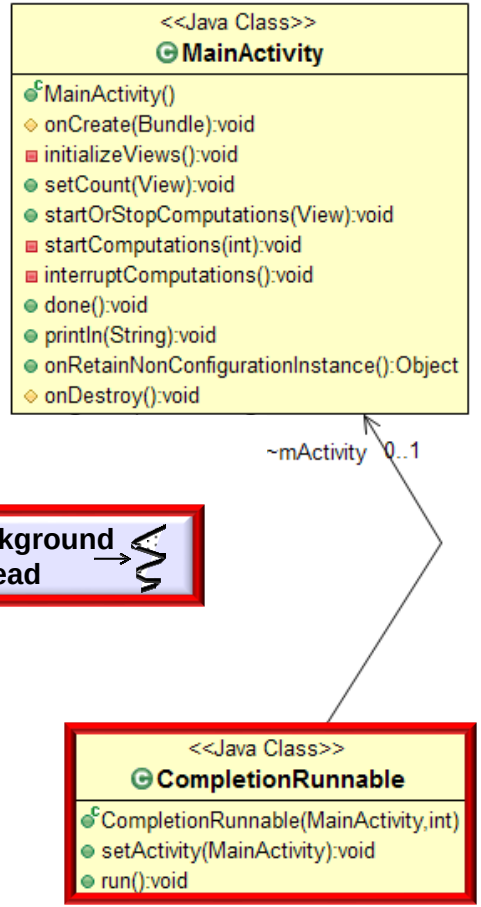
- MainActivity creates a thread to wait for all future results in the background so the UI thread doesn't block

```
...
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();
```



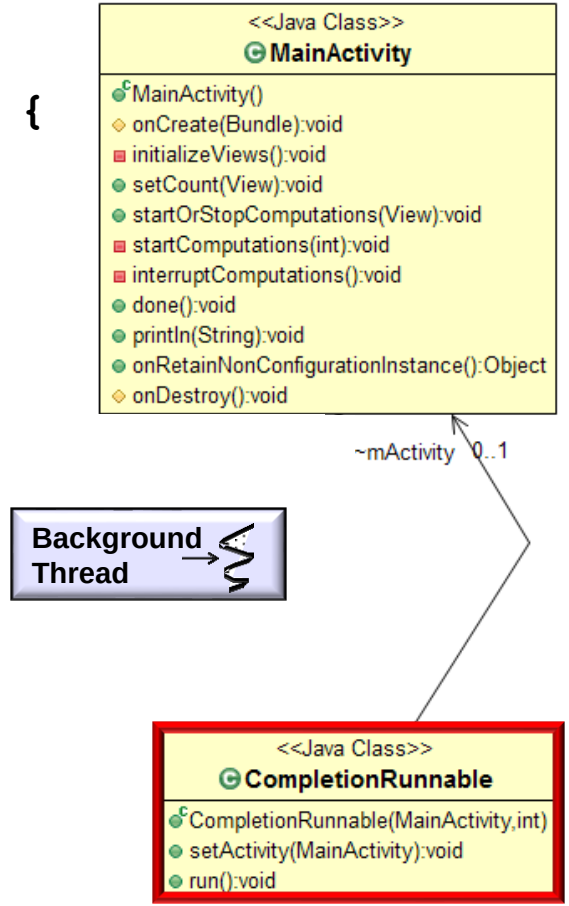
Applying TimedMemoizerEx to Check for Prime #'s

- CompletionRunnable gets results as futures complete

```
class CompletionRunnable implements Runnable {
  int mCount;
  MainActivity mActivity; ...

 public void run() {
   for (int i = 0; i < mCount; ++i) {
     PrimeResult pr = ...
       mExecutorCompService
         .take().get();

     if (pr.mSmallestFactor != 0) ...
     else ...
   ...
   mActivity.done(); ...
```



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

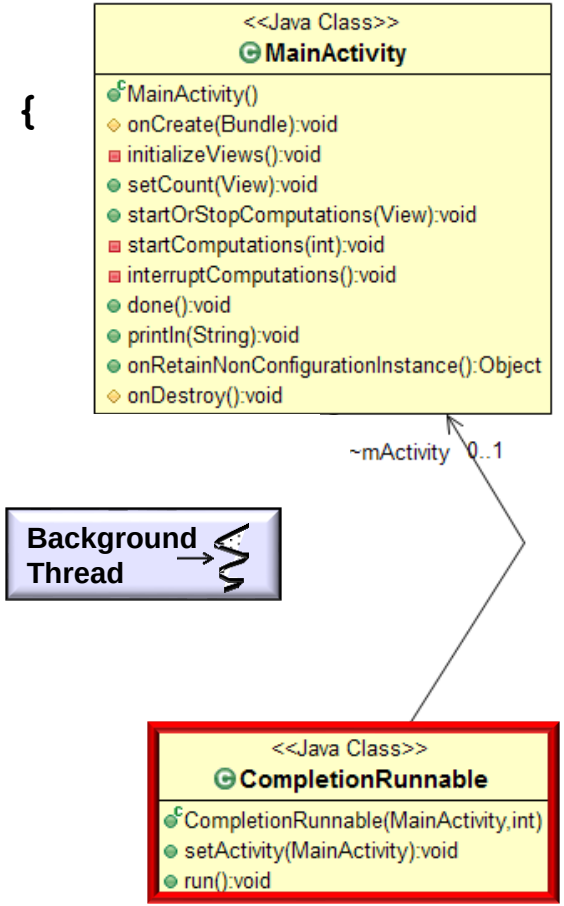
Applying TimedMemoizerEx to Check for Prime #'s

- CompletionRunnable gets results as futures complete

```
class CompletionRunnable implements Runnable {
  int mCount;
  MainActivity mActivity; ...

 public void run() {
   for (int i = 0; i < mCount; ++i) {
     PrimeResult pr = ...
       mExecutorCompService
         .take().get();

     if (pr.mSmallestFactor != 0) ...
     else ...
   ...
   mActivity.done(); ...
```



See docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html

Applying TimedMemoizerEx to Check for Prime #'s

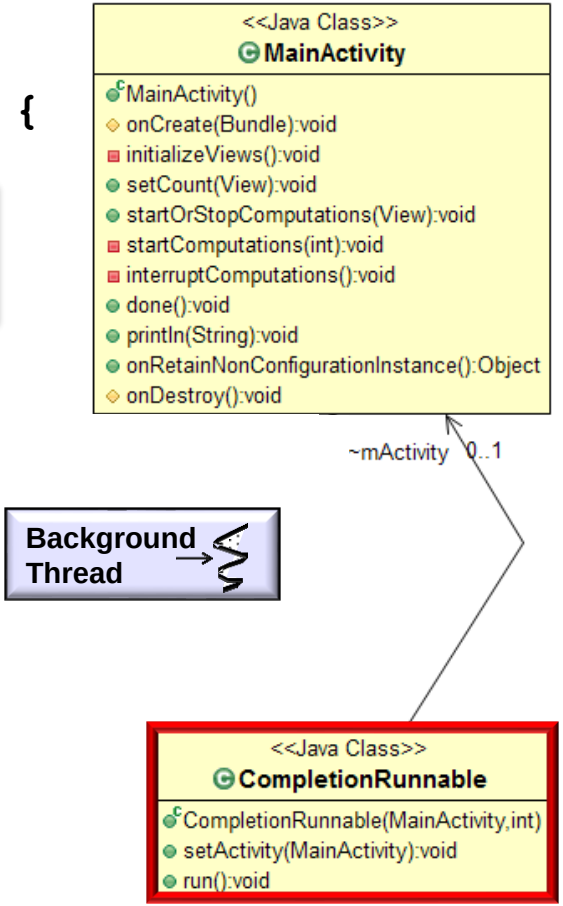
- CompletionRunnable gets results as futures complete

```
class CompletionRunnable implements Runnable {
  int mCount;
  MainActivity mActivity; ...

 public void run() {
   for (int i = 0; i < mCount; ++i) {
     PrimeResult pr = ...
       mExecutorCompService
         .take().get();

     if (pr.mSmallestFactor != 0) ...
     else ...
   ...
   mActivity.done(); ...
```

Iterate thru all results



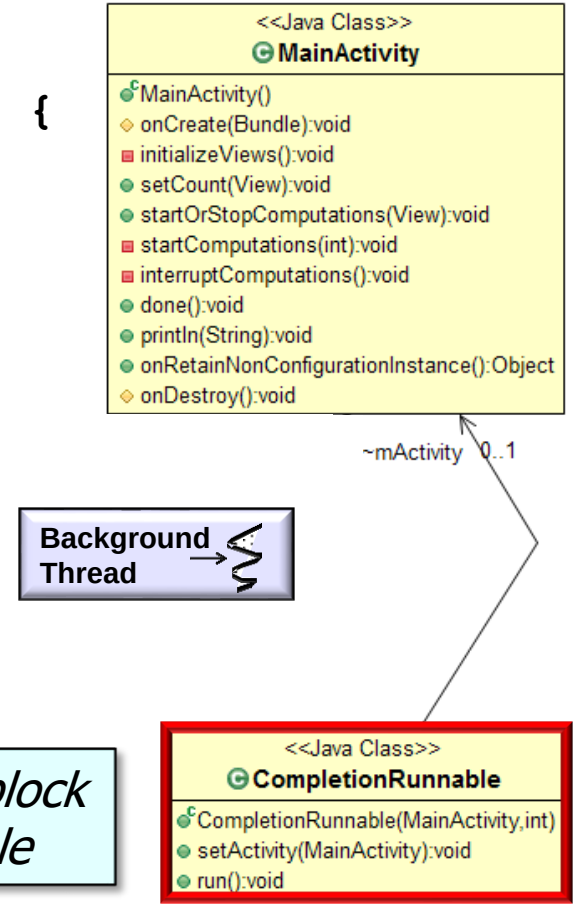
Applying TimedMemoizerEx to Check for Prime #'s

- CompletionRunnable gets results as futures complete

```
class CompletionRunnable implements Runnable {
  int mCount;
  MainActivity mActivity; ...

 public void run() {
   for (int i = 0; i < mCount; ++i) {
     PrimeResult pr = ...
       mExecutorCompService
         .take().get();

     if (pr.mSmallestFactor != 0) ...
     else ...
   ...
   mActivity.
```



get() doesn't block, though take() may block if completed futures aren't yet available

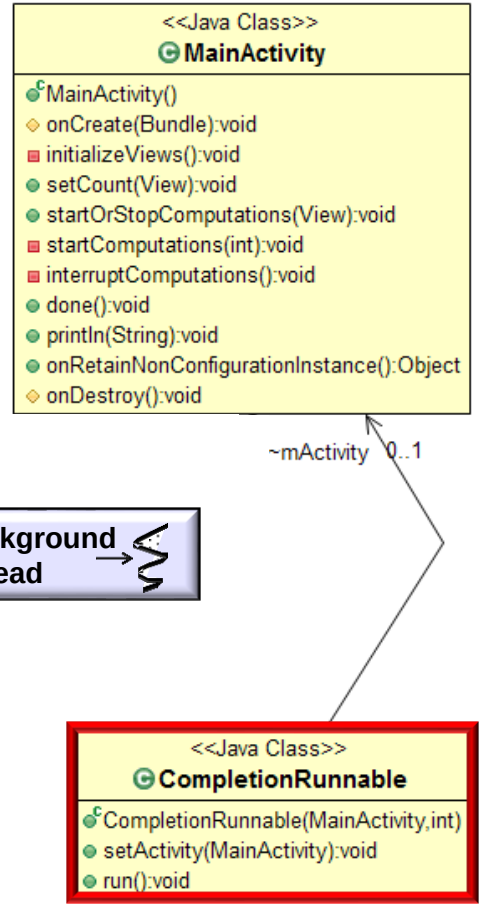
Applying TimedMemoizerEx to Check for Prime #'s

- CompletionRunnable gets results as futures complete

```
class CompletionRunnable implements Runnable {
  int mCount;
  MainActivity mActivity; ...

 public void run() {
   for (int i = 0; i < mCount; ++i) {
     PrimeResult pr = ...
       mExecutorCompService
         .take().get();

     if (pr.mSmallestFactor != 0) ...
     else ...
   ...
   mActivity.done(); ...
```



Process & output results

Applying TimedMemoizerEx to Check for Prime #'s

- RetainedState maintains key concurrency state across runtime configuration changes

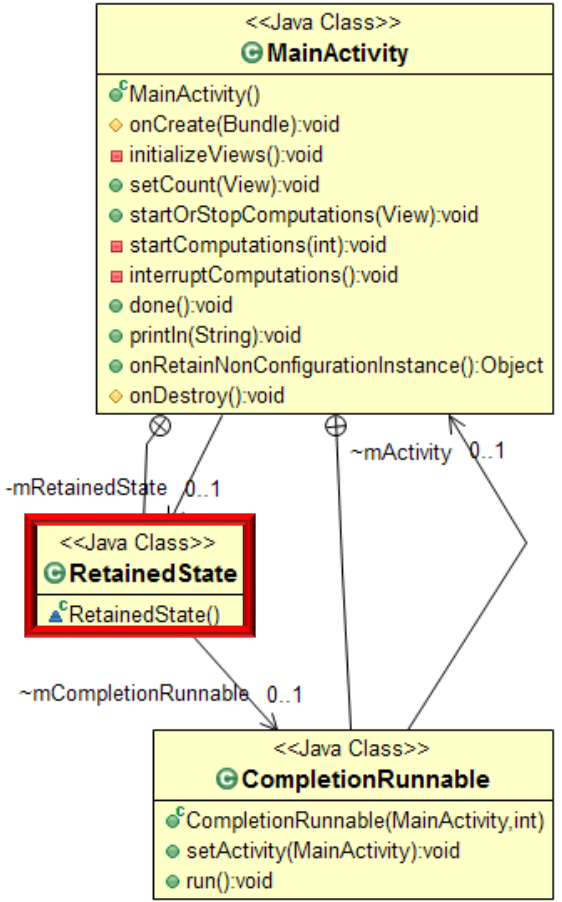
```
class RetainedState {
  ExecutorCompletionService
    mExecutorCompService;

  ExecutorService mExecutorService;

  CompletionRunnable mCompletionRunnable;

  Thread mThread;

  Memoizer<Long, Long> mMemoizer;
}
```



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

Applying TimedMemoizerEx to Check for Prime #'s

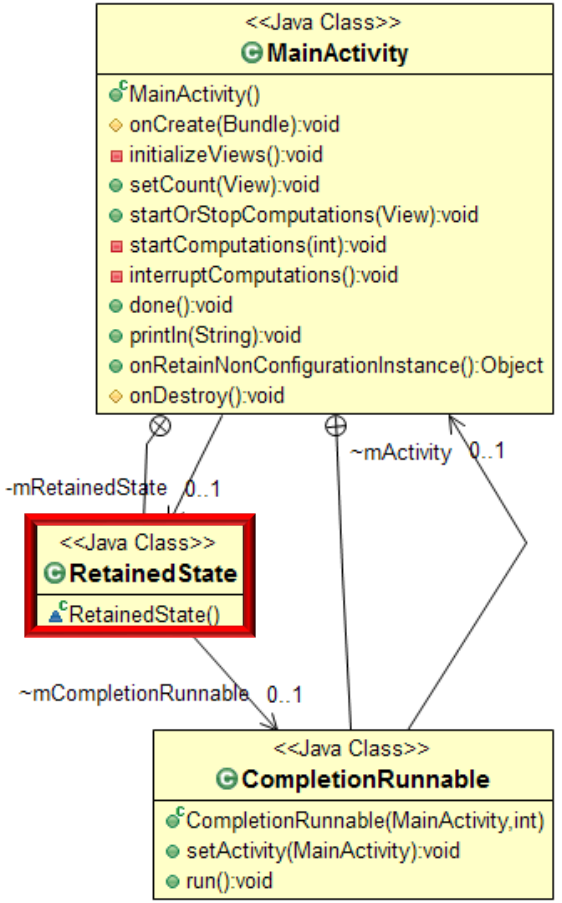
- RetainedState maintains key concurrency state across runtime configuration changes

```
void onCreate(...) {
  mRetainedState = (RetainedState)
    getLastNonConfigurationInstance();

  if (mRetainedState != null) {
    ... // update configurations
  }
}

Object onRetainNonConfigurationInstance()
{ return mRetainedState; }
```

Android's activity framework dispatches these hook methods to save & restore state when runtime configuration changes occur



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

End of Applying TimedMemoizerEx to the PrimeChecker App