Java ExecutorService: Application to PrimeChecker App

Douglas C. Schmidt

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

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

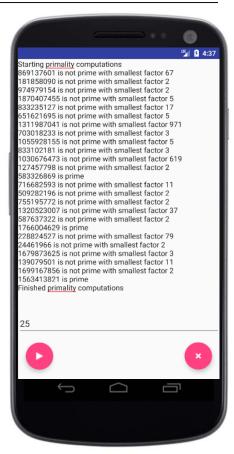
Institute for Software Integrated Systems

Vanderbilt University Nashville, Tennessee, USA



Learning Objectives in this Part of the Lesson

- Recognize the powerful features defined in the Java ExecutorService interface
- Understand other interfaces related to ExecutorService
- Know the key methods provided by ExecutorService
- Be aware of how ThreadPoolExecutor implements ExecutorService
- Learn how to program the PrimeChecker app using ExecutorService

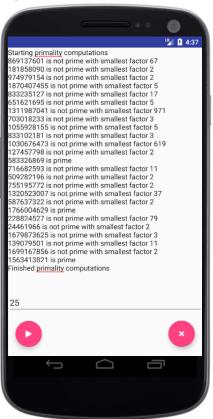


This "embarrassingly parallel" & compute-bound app uses the

Java ExecutorService to check if *N* random #'s are prime







This "embarrassingly parallel" & compute-bound app uses the

Java ExecutorService to check if N random #'s are prime

It also shows how to handle runtime configuration changes in Android



¹⁵ ₿ 4:37

Starting primality computations

69137601 is not prime with smallest factor 67 81858090 is not prime with smallest factor 2 974979154 is not prime with smallest factor 2

1870407455 is not prime with smallest factor 5 333235127 is not prime with smallest factor 17 551621695 is not prime with smallest factor 5 1311987041 is not prime with smallest factor 971 703018233 is not prime with smallest factor 3 1055928155 is not prime with smallest factor 5 33030676473 is not prime with smallest factor 3 1030676473 is not prime with smallest factor 61

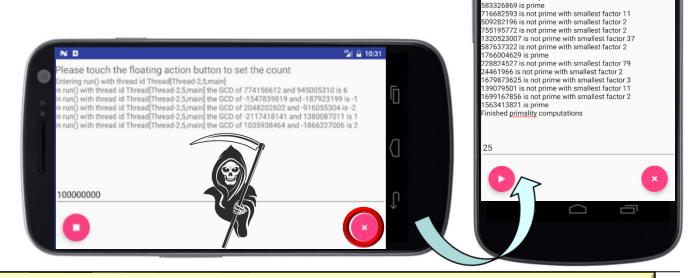
See developer.android.com/guide/topics/resources/runtime-changes.html

This "embarrassingly parallel" & compute-bound app uses the

Java ExecutorService to check if N random #'s are prime

 It also shows how to handle runtime configuration changes in Android

As well as thread interruptions



¹⁵ ₿ 4:37

Starting primality computations

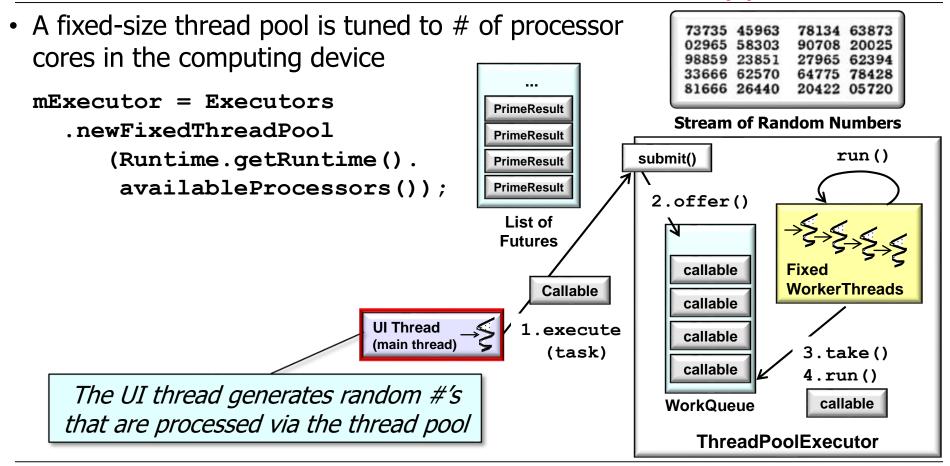
69137601 is not prime with smallest factor 67 81858090 is not prime with smallest factor 2 974979154 is not prime with smallest factor 2

1870407455 is not prime with smallest factor 5 333235127 is not prime with smallest factor 17 5311987041 is not prime with smallest factor 971 703018233 is not prime with smallest factor 971

055928155 is not prime with smallest factor 5 33102181 is not prime with smallest factor 3 030676473 is not prime with smallest factor 619 27457798 is not prime with smallest factor 2

See docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html

 A fixed-size thread pool is tuned to # of processor 73735 45963 78134 63873 02965 58303 90708 20025 cores in the computing device 98859 23851 27965 62394 33666 62570 64775 78428 81666 26440 20422 05720 mExecutor = Executors **PrimeResult** Stream of Random Numbers .newFixedThreadPool **PrimeResult** run() (Runtime.getRuntime(). submit() **PrimeResult** availableProcessors()); **PrimeResult** 2.offer() List of **Futures** callable Fixed WorkerThreads Callable callable **UI Thread** 1.execute callable (main thread) (task) 3.take() callable 4.run() WorkQueue callable **ThreadPoolExecutor**



 A fixed-size thread pool is tuned to # of processor 73735 45963 78134 63873 02965 58303 90708 20025 cores in the computing device 98859 23851 27965 62394 33666 62570 64775 78428 81666 26440 20422 05720 mExecutor = Executors **PrimeResult** Stream of Random Numbers .newFixedThreadPool **PrimeResult** run() (Runtime.getRuntime(). submit() **PrimeResult** availableProcessors()); **PrimeResult** 2.offer() List of **Futures** callable **Fixed** Callable WorkerThreads callable **UI Thread** 1.execute callable (main thread) (task) 3. take() callable 4.run() This fixed-size thread pool uses an callable WorkQueue unbounded queue to avoid deadlocks **ThreadPoolExecutor**

See <u>asznajder.github.io/thread-pool-induced-deadlocks</u>

 A fixed-size thread pool is tuned to # of processor 73735 45963 78134 63873 02965 58303 90708 20025 cores in the computing device 98859 23851 27965 62394 33666 62570 64775 78428 81666 26440 20422 05720 mExecutor = Executors**PrimeResult Stream of Random Numbers** .newFixedThreadPool **PrimeResult** run() submit() (Runtime.getRuntime(). **PrimeResult** availableProcessors()); **PrimeResult** 2.offer() List of ...mThread = new Thread(...); **Futures** callable **Fixed** ...mThread.start(); WorkerThreads Callable callable **UI Thread** 1.submit callable (main thread) (task) 3.take() Start a 2nd thread to wait callable 4.run() for the completion of all Background callable WorkQueue futures in the list of futures **Thread ThreadPoolExecutor**

This background thread ensures no blocking occurs in the UI thread

```
class PrimeCallable
                                                     <<Java Class>>
      implements Callable<PrimeResult> {
                                                    PrimeCallable
  long mPrimeCandidate;
                                                   isPrime(long):long
                                                   PrimeCallable(long)
                                                   call():PrimeResult
  PrimeCallable(Long primeCandidate)
  { mPrimeCandidate = primeCandidate; }
                                                     <<Java Class>>
  PrimeResult call() {
                                                    PrimeResult
    return new PrimeResult
                                                  (mPrimeCandidate,
                                                  isPrime(mPrimeCandidate));
                                                  PrimeResult(long,long)
```

```
class PrimeCallable
                                                     <<Java Class>>
      implements Callable<PrimeResult> {
                                                    PrimeCallable
  long mPrimeCandidate;
                                                    isPrime(long):long
                             Implements Callable
                                                    PrimeCallable(long)
                                                    call():PrimeResult
  PrimeCallable(Long primeCandidate)
   mPrimeCandidate = primeCandidate; }
                                                     <<Java Class>>
  PrimeResult call() {
                                                     PrimeResult
    return new PrimeResult
                                                  (mPrimeCandidate,
                                                  isPrime(mPrimeCandidate));
                                                  PrimeResult(long,long)
```

```
class PrimeCallable
                                                      <<Java Class>>
      implements Callable<PrimeResult> {
                                                      PrimeCallable
  long mPrimeCandidate;
                           Constructor stores prime
                                                     ■ isPrime(long):long
                                                     PrimeCallable(long)
                             # candidate in a field
                                                     call():PrimeResult
  PrimeCallable(Long primeCandidate)
  { mPrimeCandidate = primeCandidate; }
                                                      <<Java Class>>
  PrimeResult call() {
                                                      PrimeResult
    return new PrimeResult
                                                    (mPrimeCandidate,
                                                    isPrime(mPrimeCandidate));
                                                   PrimeResult(long,long)
```

PrimeCallable defines a two-way means of determining whether a # is prime class PrimeCallable

```
<<Java Class>>
    implements Callable<PrimeResult> {
                                                   PrimeCallable
long mPrimeCandidate;
                                                  ■ isPrime(long):long
                                                  PrimeCallable(long)
                                                  call():PrimeResult
PrimeCallable(Long primeCandidate)
 mPrimeCandidate = primeCandidate; }
                                                   <<Java Class>>
PrimeResult call() {
                                                   PrimeResult
  return new PrimeResult
                                                 (mPrimeCandidate,
                                                 isPrime (mPrimeCandidate) );
                                                PrimeResult(long,long)
```

call() hook method invokes isPrime() in a pool thread

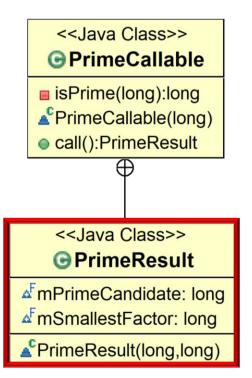
Interruptible isPrime() based on "Java Executor: Application to PrimeChecker App"

PrimeCallable defines a two-way means of determining whether a # is prime

```
class PrimeCallable
                                                        <<Java Class>>
      implements Callable<PrimeResult> {
                                                       PrimeCallable
  long mPrimeCandidate;
                                                       isPrime(long):long
                                                       PrimeCallable(long)
                                                       call():PrimeResult
   PrimeResult is a tuple that matches the prime #
   candidate with the result of checking primality
                                                        <<Java Class>>
  PrimeResult call() {
                                                        PrimeResult
    return new PrimeResult
                                                     (mPrimeCandidate,
                                                     isPrime(mPrimeCandidate));
                                                     PrimeResult(long,long)
```

These two-way semantics eliminate the need for a dependency on MainActivity!

```
class PrimeCallable
      implements Callable<PrimeResult> {
  long mPrimeCandidate;
  PrimeCallable(Long primeCandidate)
   mPrimeCandidate = primeCandidate; }
  PrimeResult call() {
    return new PrimeResult
       (mPrimeCandidate,
       isPrime(mPrimeCandidate));
     These two-way call semantics eliminate the
     need for any dependency on MainActivity!
```



```
class PrimeCallable
                                                        <<Java Class>>
      implements Callable<PrimeResult> {
                                                       PrimeCallable
                              Returns 0 if n is prime or
                                                      ■ isPrime(long):long
                                                      PrimeCallable(long)
                              smallest factor if it's not
  long isPrime(long n) {
                                                      call():PrimeResult
    if (n > 3)
      for (long factor = 2;
            factor <= n / 2; ++factor)</pre>
                                                        <<Java Class>>
         if (Thread.interrupted()) break;
                                                        PrimeResult
         else if (n / factor * factor == n)
                                                     return factor;
                                                     return OL;
                                                     PrimeResult(long,long)
```

PrimeCallable defines a two-way means of determining whether a # is prime class PrimeCallable

```
<<Java Class>>
    implements Callable<PrimeResult> {
                                                     PrimeCallable
                                                    isPrime(long):long
                                                    PrimeCallable(long)
long isPrime(long n) {
                                                    call():PrimeResult
  if (n > 3)
    for (long factor = 2;
          factor <= n / 2; ++factor)</pre>
                                                     <<Java Class>>
      if (Thread.interrupted()) break;
                                                     PrimeResult
      else if (n / factor | * factor == n)
                                                   return factor;
                                                   return OL;
                                                   PrimeResult(long,long)
                      isPrime() repeatedly checks
                      to see if it's been interrupted
```

See lesson on "Managing the Java Thread Lifecycle"

PrimeResult

PrimeResult

PrimeResult

PrimeResult

List of

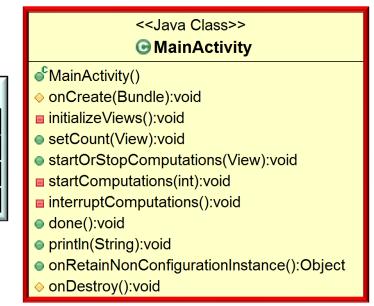
Futures

 MainActivity creates a list of futures that store results of concurrently checking primality of "count" random #'s within a range

List<Future<PrimeResult>>

futures = ...

This list of futures is initialized via a Java sequential stream



 MainActivity creates a list of futures that store <<Java Class>> results of concurrently checking primality of MainActivity MainActivity() "count" random #'s within a range onCreate(Bundle):void initializeViews():void PrimeResult List<Future<PrimeResult>> setCount(View):void **PrimeResult** futures = new Random() startOrStopComputations(View):void **PrimeResult** startComputations(int):void .longs(count, **PrimeResult** interruptComputations():void **SMAX VALUE** count,

List of

Futures

odone():void

println(String):void

onDestroy():void

onRetainNonConfigurationInstance():Object

Generates "count" random #'s ranging from sMAX_VALUE - count & sMAX_VALUE

sMAX VALUE)

 MainActivity creates a list of futures that store <<Java Class>> results of concurrently checking primality of MainActivity "count" random #'s within a range onCreate(Bundle):void initializeViews():void **PrimeResult** List<Future<PrimeResult>> setCount(View):void **PrimeResult** futures = new Random() startOrStopComputations(View):void **PrimeResult** startComputations(int):void .longs(count, **PrimeResult** interruptComputations():void sMAX VALUE - count, odone():void List of **SMAX VALUE)** println(String):void **Futures** onRetainNonConfigurationInstance():Object onDestroy():void .mapToObj(PrimeCallable::new)

This constructor reference converts random #'s into PrimeCallables

 MainActivity creates a list of futures that store <<Java Class>> results of concurrently checking primality of MainActivity "count" random #'s within a range onCreate(Bundle):void initializeViews():void **PrimeResult** List<Future<PrimeResult>> setCount(View):void **PrimeResult** futures = new Random() startOrStopComputations(View):void **PrimeResult** startComputations(int):void .longs(count, **PrimeResult** interruptComputations():void sMAX VALUE - count, odone():void List of **SMAX VALUE)** println(String):void **Futures** onRetainNonConfigurationInstance():Object onDestroy():void .mapToObj(PrimeCallable::new)

.map (mRetainedState.mExecutorService::submit)

Submit a two-way task for execution & return a future representing pending task results

 MainActivity creates a list of futures that store <<Java Class>> results of concurrently checking primality of MainActivity MainActivity() "count" random #'s within a range onCreate(Bundle):void initializeViews():void **PrimeResult** List<Future<PrimeResult>> setCount(View):void **PrimeResult** futures = new Random() startOrStopComputations(View):void **PrimeResult** startComputations(int):void .longs(count, **PrimeResult** interruptComputations():void sMAX VALUE - count, odone():void

.mapToObj(PrimeCallable::new)

SMAX VALUE)

.map (mRetainedState.mExecutorService::submit)

.collect(toList()); Collect results into a list of futures to PrimeResults

List of

Futures

println(String):void

onDestroy():void

onRetainNonConfigurationInstance():Object

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#collect

<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void class FutureRunnable implements Runnable { startOrStopComputations(View):void startComputations(int):void List<Future<PrimeResult>> interruptComputations():void odone():void mFutures; println(String):void onRetainNonConfigurationInstance():Object onDestroy():void ~mActivity MainActivity mActivity; Background 4 **Thread** FutureRunnable (MainActivity a, List<Future<PrimeResult>> f) <<Java Class>> FutureRunnable mActivity = a; mFutures = f; } FutureRunnable(MainActivity,List<Future<PrimeResult>>) setActivity(MainActivity):void o run():void

<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void class FutureRunnable implements Runnable { startOrStopComputations(View):void startComputations(int):void List<Future<PrimeResult>> interruptComputations():void odone():void mFutures; println(String):void onRetainNonConfigurationInstance():Object onDestroy():void ~mActivity MainActivity mActivity; Background 4 **Thread** FutureRunnable (MainActivity a, List<Future<PrimeResult>> f) <<Java Class>> FutureRunnable mActivity = a; mFutures = f; } FutureRunnable(MainActivity,List<Future<PrimeResult>>) setActivity(MainActivity):void o run():void

<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void class FutureRunnable implements Runnable { startOrStopComputations(View):void startComputations(int):void List<Future<PrimeResult>> interruptComputations():void odone():void mFutures; println(String):void List of futures to results of onRetainNonConfigurationInstance():Object onDestroy():void PrimeCallable computations ~mActivity MainActivity mActivity; Background & **Thread** FutureRunnable (MainActivity a, List<Future<PrimeResult>> f) <<Java Class>> **⊕** FutureRunnable mActivity = a; mFutures = f; } FutureRunnable(MainActivity,List<Future<PrimeResult>>) setActivity(MainActivity):void o run():void

<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void class FutureRunnable implements Runnable { startOrStopComputations(View):void startComputations(int):void List<Future<PrimeResult>> interruptComputations():void odone():void mFutures; println(String):void Reference back to enclosing activity onRetainNonConfigurationInstance():Object onDestroy():void ~mActivity MainActivity mActivity; Background 4 **Thread** FutureRunnable (MainActivity a, List<Future<PrimeResult>> f) <<Java Class>> **⊕** FutureRunnable mActivity = a; mFutures = f; } FutureRunnable(MainActivity,List<Future<PrimeResult>>) setActivity(MainActivity):void o run():void

<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void class FutureRunnable implements Runnable { startOrStopComputations(View):void startComputations(int):void List<Future<PrimeResult>> interruptComputations():void odone():void mFutures; println(String):void onRetainNonConfigurationInstance():Object onDestroy():void ~mActivity MainActivity mActivity; Background 4 Constructor initializes the fields **Thread** FutureRunnable (MainActivity a, List<Future<PrimeResult>> f) <<Java Class>> **⊕** FutureRunnable mActivity = a; mFutures = f; } FutureRunnable(MainActivity,List<Future<PrimeResult>>) setActivity(MainActivity):void o run():void

<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void Runnable hook method public void run() { startOrStopComputations(View):void startComputations(int):void mFutures.forEach(future -> { interruptComputations():void odone():void PrimeCallable.PrimeResult pr = println(String):void onRetainNonConfigurationInstance():Object rethrowSupplier(future::get).get(); onDestroy():void ~mActivity (pr.mSmallestFactor != 0) Background & **Thread** else ... }); <<Java Class>> FutureRunnable ▲ FutureRunnable(MainActivity, List<Future<PrimeResult>>) mActivity.done(); ... setActivity(MainActivity):void o run():void

<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void Iterate thru all futures public void run() startOrStopComputations(View):void startComputations(int):void mFutures.forEach(future -> { interruptComputations():void odone():void PrimeCallable.PrimeResult pr = println(String):void onRetainNonConfigurationInstance():Object rethrowSupplier(future::get).get(); onDestroy():void ~mActivity (pr.mSmallestFactor != 0) Background & **Thread** else ... }); <<Java Class>> FutureRunnable ▲ FutureRunnable(MainActivity, List<Future<PrimeResult>>) mActivity.done(); ... setActivity(MainActivity):void o run():void

<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void public void run() { startOrStopComputations(View):void startComputations(int):void mFutures.forEach(future -> { interruptComputations():void odone():void PrimeCallable.PrimeResult pr = println(String):void onRetainNonConfigurationInstance():Object rethrowSupplier(future::get).get(); onDestroy():void ~mActivity if (pr.mSmallestFactor != 0) Background_< **Thread** else ...}); future::get blocks if async processing <<Java Class>> associated with future hasn't completed FutureRunnable ▲ FutureRunnable(MainActivity, List<Future<PrimeResult>>) mActivity.done(); ... setActivity(MainActivity):void run():void

This is an example of the "synchronous future" processing model

<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void public void run() { startOrStopComputations(View):void startComputations(int):void mFutures.forEach(future -> { interruptComputations():void odone():void PrimeCallable.PrimeResult pr = println(String):void onRetainNonConfigurationInstance():Object rethrowSupplier(future::get).get(); onDestroy():void ~mActivity if (pr.mSmallestFactor != 0) Background 4 **Thread** Convert checked exception else ...}); to a runtime exception <<Java Class>> FutureRunnable ▲ FutureRunnable(MainActivity, List<Future<PrimeResult>>) mActivity.done(); ... setActivity(MainActivity):void run():void

See stackoverflow.com/a/27644392/3312330

<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void public void run() { startOrStopComputations(View):void startComputations(int):void mFutures.forEach(future -> { interruptComputations():void odone():void PrimeCallable.PrimeResult pr = println(String):void onRetainNonConfigurationInstance():Object rethrowSupplier(future::get).get(); onDestroy():void ~mActivity (pr.mSmallestFactor != 0) Background_< **Thread** else ... }); Get the result from the supplier <<Java Class>> FutureRunnable ▲ FutureRunnable(MainActivity, List<Future<PrimeResult>>) mActivity.done(); ... setActivity(MainActivity):void run():void

See docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html#get

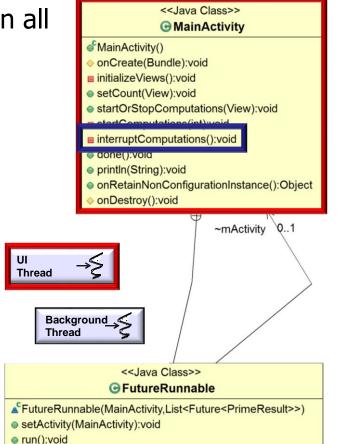
<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void public void run() { startOrStopComputations(View):void startComputations(int):void mFutures.forEach(future -> { interruptComputations():void odone():void PrimeCallable.PrimeResult pr = println(String):void onRetainNonConfigurationInstance():Object rethrowSupplier(future::get).get(); onDestroy():void ~mActivity if (pr.mSmallestFactor != 0) Background 4 **Thread** else ...}); Process each result & produce output <<Java Class>> FutureRunnable ▲ FutureRunnable(MainActivity, List<Future<PrimeResult>>) mActivity.done(); ... setActivity(MainActivity):void o run():void

<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void public void run() { startOrStopComputations(View):void startComputations(int):void mFutures.forEach(future -> { interruptComputations():void odone():void PrimeCallable.PrimeResult pr = println(String):void onRetainNonConfigurationInstance():Object rethrowSupplier(future::get).get(); onDestroy():void ~mActivity if (pr.mSmallestFactor != 0) Background 4 **Thread** else ...}); Inform MainActivity that we're all done <<Java Class>> FutureRunnable ▲ FutureRunnable(MainActivity, List<Future<PrimeResult>>) mActivity.done(); setActivity(MainActivity):void o run():void

 The interruptComputations() method shuts down all the concurrent computations via the UI thread

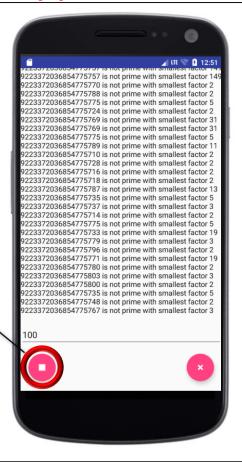
```
mRetainedState.mThread.interrupt();
...
```

```
mRetainedState
   .mExecutorService.awaitTermination
      (500, TimeUnit.MILLISECONDS);
```



 The interruptComputations() method shuts down all the concurrent computations via the UI thread

```
void interruptComputations()
  mRetainedState\mExecutorService
                  shutdownNow();
       Called when user presses the cancel button
  mRetainedState.mThread.interrupt();
  mRetainedState
    .mExecutorService.awaitTermination
      (500, TimeUnit.MILLISECONDS);
```



 The interruptComputations() method shuts down all the concurrent computations via the UI thread

void interruptComputations() mRetainedState.mExecutorService <. shutdownNow();</pre>

> Abruptly shutdown the executor service, which interrupts all threads running tasks

mRetainedState.mThread.interrupt();

mRetainedState

.mExecutorService.awaitTermination

(500, TimeUnit.MILLISECONDS);

<<Java Class>> MainActivity MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void startOrStopComputations(View):void interruptComputations():void aone():voia println(String):void onRetainNonConfigurationInstance():Object onDestroy():void ~mActivity 0..1 **Thread** Background , <<Java Class>> FutureRunnable ▲ FutureRunnable(MainActivity,List<Future<PrimeResult>>) setActivity(MainActivity):void

See docs.orade.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#shutdownNow

o run():void

 The interruptComputations() method shuts down all the concurrent computations via the UI thread

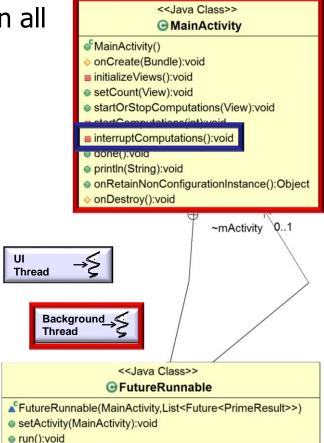
```
mRetainedState.mThread.interrupt();
```

Interrupt the background thread

mRetainedState

.mExecutorService.awaitTermination

(500, TimeUnit.MILLISECONDS);



See docs.oracle.com/javase/8/docs/api/java/lang/Thread.html#interrupt

 The interruptComputations() method shuts down all the concurrent computations via the UI thread

void interruptComputations() mRetainedState.mExecutorService .shutdownNow();

Block until all tasks have completed execution

mRetainedState.mThread.interrupt();

mRetainedState

.mExecutorService.awaitTermination

(500, TimeUnit.MILLISECONDS);

MainActivity MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void startOrStopComputations(View):void interruptComputations():void aone():voia println(String):void onRetainNonConfigurationInstance():Object onDestroy():void ~mActivity 0..1 Thread Background < Thread <<Java Class>> FutureRunnable ▲ FutureRunnable(MainActivity,List<Future<PrimeResult>>) setActivity(MainActivity):void o run():void

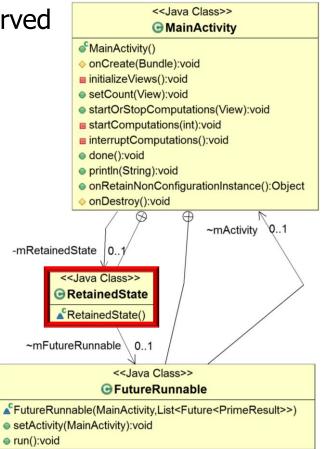
<<Java Class>>

See docs.orade.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html#awaitTermination

 RetainedState contains fields that must be preserved across Android runtime configuration changes class RetainedState {

```
ExecutorService mExecutorService;
FutureRunnable mFutureRunnable;
Thread mThread;
```

These fields store concurrency-related objects

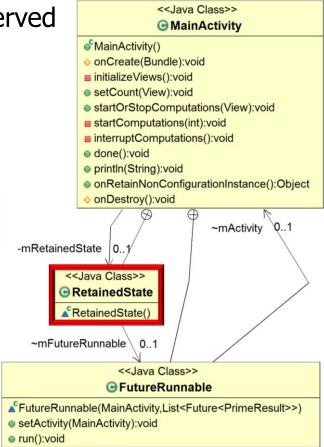


 RetainedState contains fields that must be preserved across Android runtime configuration changes

```
mRetainedState.mFutureRunnable =
  new FutureRunnable(this, futures);
```

FutureRunnable is stored in a field so its state can be updated during a runtime configuration change

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



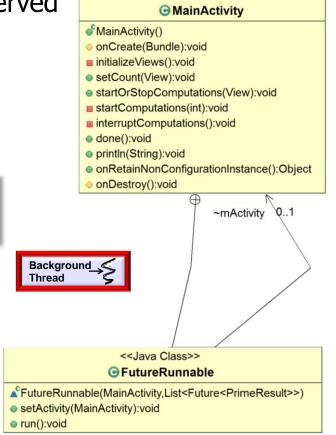
See developer.android.com/guide/topics/resources/runtime-changes.html

 RetainedState contains fields that must be preserved across Android runtime configuration changes
 ...

```
mRetainedState.mFutureRunnable =
  new FutureRunnable(this, futures);
```

A background thread is started to wait for all future results to avoid blocking the UI thread

mRetainedState.mThread.start();



<<Java Class>>

See developer.android.com/training/articles/perf-anr.html

<<Java Class>> Android provides hook methods to store & retrieve MainActivity app state across runtime configuration changes MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void startOrStopComputations(View):void Object onRetainNonConfigurationInstance() startComputations(int):void interruptComputations():void return mRetainedState; } o done():void println(String):void onRetainNonConfigurationInstance():Object Retained state is loaded/stored onDestroy():void via Android hook methods ~mActivity -mRetainedState void onCreate(...) { <<Java Class>> mRetainedState = (RetainedState) RetainedState RetainedState() getLastNonConfigurationInstance(); ~mFutureRunnable \ 0..1 <<Java Class>> (mRetainedState != null) { FutureRunnable ▲ FutureRunnable(MainActivity,List<Future<PrimeResult>>) setActivity(MainActivity):void run():void

See developer.android.com/reference/android/app/Activity.html#onRetainNonConfigurationInstance()

End of Java Executor Service: Application to PrimeChecker App