The Java ExecutorService Interface (Part 4)

Douglas C. Schmidt

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

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

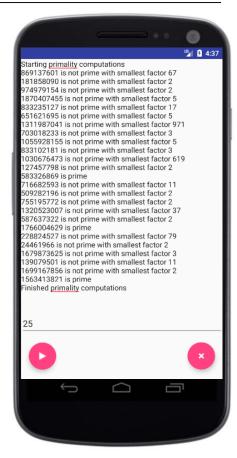
Institute for Software Integrated Systems

Vanderbilt University Nashville, Tennessee, USA



Learning Objectives in this Part of the Lesson

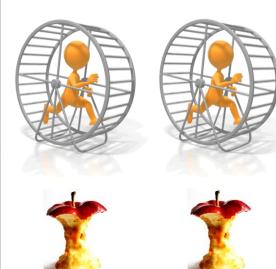
- Recognize the powerful features defined in the Java ExecutorService interface & related interfaces/classes
- Know the key methods provided by the Java ExecutorService
- Understand how ThreadPoolExecutor implements the ExecutorService
- Learn how to program a "PrimeChecker" app using the Java ExecutorService interface

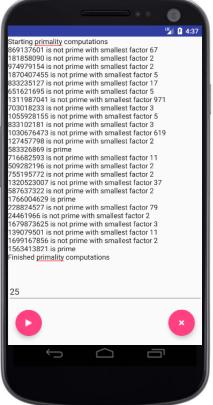


This "embarrassingly parallel" & compute-bound app uses the

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







- This "embarrassingly parallel" app shows how the Java ExecutorService can determine if N random #'s are prime
 - It also shows how to handle runtime configuration changes in Android



IE 4:37

tarting primality computations

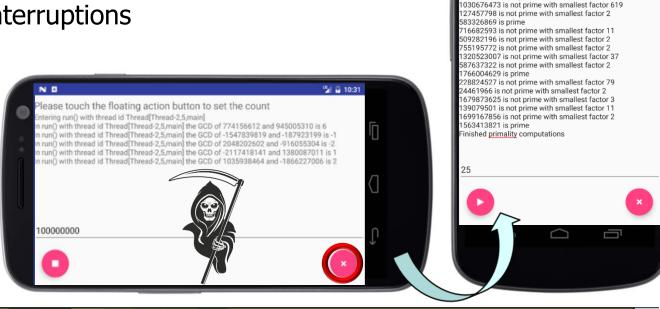
8691376<mark>01 is not prime with smallest factor 67</mark> 181858090 is not prime with smallest factor 2

974979154 is not prime with smallest factor 2 1870407455 is not prime with smallest factor 5 833235127 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

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

- This "embarrassingly parallel" app shows how the Java ExecutorService can determine if N random #'s are prime
 - It also shows how to handle runtime configuration changes in Android
 - As well as thread interruptions



tarting primality computations

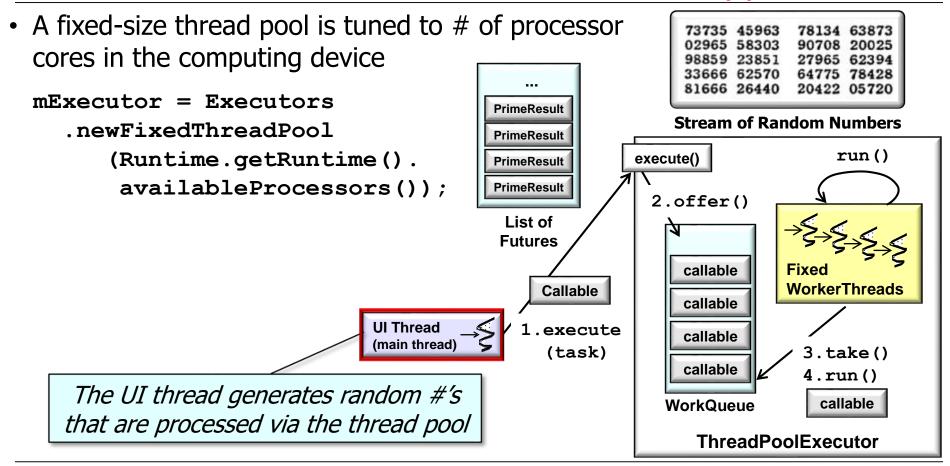
8691376<mark>01 is not prime with smallest factor 67</mark> 181858090 is not prime with smallest factor 2

974979154 is not prime with smallest factor 2 1870407455 is not prime with smallest factor 5 833235127 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 833102181 is not prime with smallest factor 3

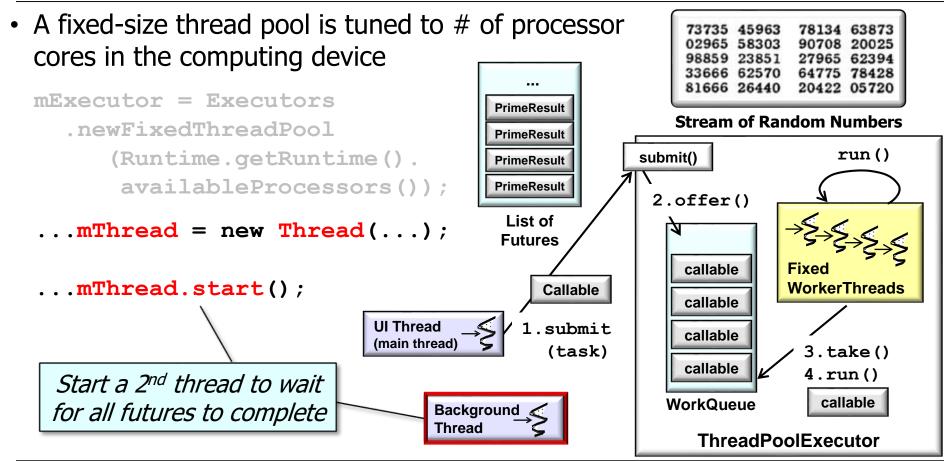
See docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html

 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(). execute() **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() execute() (Runtime.getRuntime(). **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>



 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>>
long isPrime(long n) { ... }
                                                     PrimeResult
```

PrimeResult call() { return new PrimeResult (mPrimeCandidate, PrimeResult(long,long) isPrime(mPrimeCandidate));

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

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

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

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

```
class PrimeCallable
                                                      <<Java Class>>
      implements Callable<PrimeResult> {
                                                     PrimeCallable
  long mPrimeCandidate;
                           The constructor stores
                                                    ■ isPrime(long):long
                                                    PrimeCallable(long)
                           the prime # candidate
                                                    call():PrimeResult
  PrimeCallable(Long primeCandidate)
  { mPrimeCandidate = primeCandidate; }
                                                      <<Java Class>>
  long isPrime(long n) { ... }
                                                     PrimeResult
                                                   PrimeResult call() {
                                                   return new PrimeResult (mPrimeCandidate,
                                                   PrimeResult(long,long)
       isPrime(mPrimeCandidate));
```

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>>
                         Returns 0 if n is prime or
long isPrime(long n)
                                                    PrimeResult
                         smallest factor if it's not
                                                  PrimeResult call() {
                                                  return new PrimeResult (mPrimeCandidate,
                                                  PrimeResult(long,long)
     isPrime(mPrimeCandidate));
```

An interruptible version of isPrime() from "The Java Executor Interface (Part 2)"

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>>
                           The call() hook method
  long isPrime(long n)
                                                      PrimeResult
                             invokes isPrime()
                                                   PrimeResult call()
                                                   return new PrimeResult (mPrimeCandidate,
                                                   PrimeResult(long,long)
       isPrime(mPrimeCandidate));
```

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>>
  long isPrime(long n) { ... }
                                                     PrimeResult
                                                   PrimeResult call() {
                                                   return new PrimeResult (mPrimeCandidate,
                                                   PrimeResult(long,long)
       isPrime(mPrimeCandidate));
           PrimeResult is a tuple that matches the prime #
```

candidate with the result of checking primality

PrimeResult

PrimeResult

PrimeResult

PrimeResult

List of

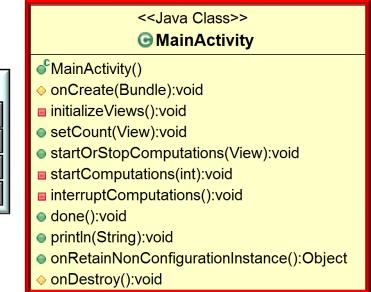
Futures

 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 8 sequential stream



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

<<Java Class>> MainActivity 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

Generates "count" random #'s ranging from sMAX_VALUE - count & 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>> MainActivity 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() **PrimeResult** .longs(count, **PrimeResult** sMAX VALUE - count, List of

.mapToObj(PrimeCallable::new)

SMAX VALUE)

MainActivity()
 onCreate(Bundle):void
 initializeViews():void
 setCount(View):void
 startOrStopComputations(View):void
 startComputations(int):void
 interruptComputations():void
 done():void
 println(String):void
 onRetainNonConfigurationInstance():Object
 onDestroy():void

.map (mRetainedState.mExecutorService::submit)

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

Futures

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 onDestrov():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

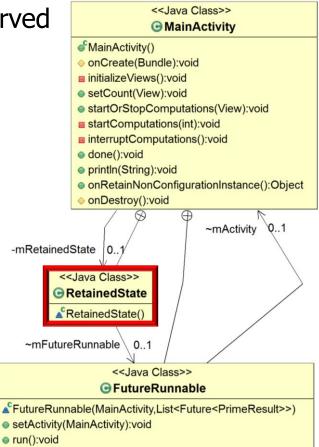
<<Java Class>> FutureRunnable runs in a background thread & gets MainActivity the results of all futures as they complete MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void public void run() { startOrStopComputations(View):void startComputations(int):void mFutures.forEach(future -> { 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

 RetainedState contains fields that must be preserved across runtime configuration changes

```
class RetainedState {
   ExecutorService mExecutorService;
   FutureRunnable mFutureRunnable;
   Thread mThread;
}
```

These fields store concurrency-related objects

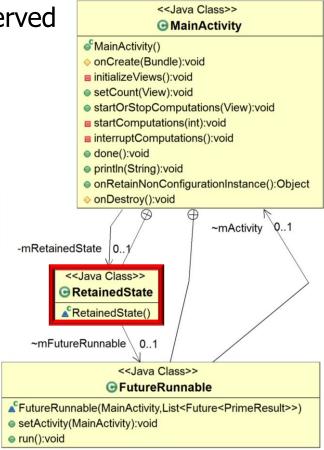


 RetainedState contains fields that must be preserved across runtime configuration changes

```
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

<<Java Class>> RetainedState contains fields that must be preserved MainActivity across runtime configuration changes MainActivity() onCreate(Bundle):void initializeViews():void setCount(View):void mRetainedState.mFutureRunnable = startOrStopComputations(View):void startComputations(int):void new FutureRunnable(this, futures); interruptComputations():void odone():void println(String):void onRetainNonConfigurationInstance():Object onDestroy():void A background thread is started to wait for all ~mActivity future results to avoid blocking the UI thread Background & **Thread** mRetainedState.mThread = new Thread (mRetainedState .mFutureRunnable); <<Java Class>>

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

mRetainedState.mThread.start();

⊚ FutureRunnable

△ FutureRunnable(MainActivity,List<Future<PrimeResult>>)

setActivity(MainActivity):void

run():void

<<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; } odone():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

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

run():void

 ExecutorService version of PrimeChecker app fixes problems with earlier Executor PrimeChecker



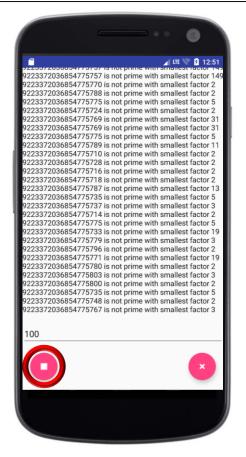
- ExecutorService version of PrimeChecker app fixes
- problems with earlier Executor PrimeChecker, e.g.Two-way semantics of Java callables decouple

PrimeCallable & MainActivity



This decoupling simplifies runtime configuration changes

- ExecutorService version of PrimeChecker app fixes problems with earlier Executor PrimeChecker, e.g.
 - Two-way semantics of Java callables decouple
 PrimeCallable & MainActivity
 - Lifecycle operations enable task interruptions



Shutting down an executor service interrupts all threads running tasks

- ExecutorService version of PrimeChecker app fixes problems with earlier Executor PrimeChecker, e.g.
 - Two-way semantics of Java callables decouple
 PrimeCallable & MainActivity
 - Lifecycle operations enable task interruptions

```
long isPrime(long n) {
  if (n > 3)
    for (long factor = 2;
        factor <= n / 2; ++factor)
    if (Thread.interrupted()) break;
    else if (n / factor * factor == n)
        return factor;
  return OL;
}</pre>
```



The isPrime() method repeatedly checks to see if it's been interrupted

- ExecutorService version of PrimeChecker app fixes problems with earlier Executor PrimeChecker, e.g.
 - Two-way semantics of Java callables decouple
 PrimeCallable & MainActivity
 - Lifecycle operations enable task interruptions
 - Runtime configuration changes handled gracefully





Starting primality computations 369137601 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 33235127 is not prime with smallest factor 17 51621695 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 33102181 is not prime with smallest factor 3 1030676473 is not prime with smallest factor 619 127457798 is not prime with smallest factor 2 716682593 is not prime with smallest factor 11 509282196 is not prime with smallest factor 2 755195772 is not prime with smallest factor 2 1320523007 is not prime with smallest factor 37 587637322 is not prime with smallest factor 2 1766004629 is prime 228824527 is not prime with smallest factor 79 24461966 is not prime with smallest factor 2 1679873625 is not prime with smallest factor 3 139079501 is not prime with smallest factor 11 1699167856 is not prime with smallest factor 2 1563413821 is prime Finished primality computations

Running tasks execute & update the GUI until they finish or are interrupted

However, there are still some limitations



- However, there are still some limitations, e.g.
 - future::get blocks the thread, even if other futures may have completed private class FutureRunnable

```
implements Runnable {
                                        This problem is inherent with the
```

```
MainActivity mActivity; ...
                                    "synchronous future" processing model
public void run() {
```

```
mFutures.forEach(future -> {
  PrimeCallable.PrimeResult pr =
    rethrowSupplier(future::get).get();
if (pr.mSmallestFactor != 0) ...
else ...
```

mActivity.done(); ...

We fix this problem in an upcoming lesson on "Java ExecutorCompletionService"!

However, there are still some limitations, e.g.

return OL;

- future::get blocks the thread, even if other futures may have completed
- isPrime() tightly coupled with PrimeCallable Primality checking always runs, even if results were public class PrimeCallable ... { long isPrime(long n) { if (n > 3)for (long factor = 2; factor <= n / 2; ++factor) if (Thread.interrupted()) break; else if (n / factor * factor == n) return factor;

computed previously

This problem is fixed by Memoizer in an upcoming lesson on "Java FutureTask"!

End of Overview of Java ExecutorService (Part 4)