# **The Java Executor Interface (Part 2)**

Douglas C. Schmidt

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

www.dre.vanderbilt.edu/~schmidt



**Professor of Computer Science** 

**Institute for Software Integrated Systems** 

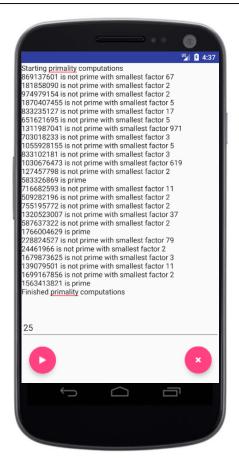
Vanderbilt University Nashville, Tennessee, USA



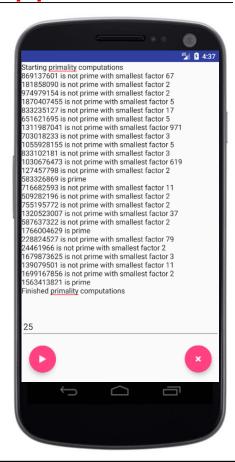
#### Learning Objectives in this Part of the Lesson

- Recognize the simple/single feature provided by the Java Executor interface
- Learn how to program a simple "prime checker" app using the Java Executor interface



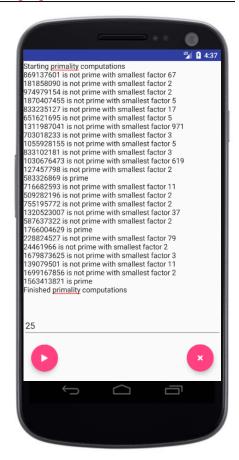


 This app shows how to use the Java Executor framework to check if N random #'s are prime



- This app shows how to use the Java Executor framework to check if N random #'s are prime
  - Each natural # divisible only by 1 & itself is prime

2	3	5	7	11
13	17	19	23	29
31	37	41	43	47
53	59	61	67	71
73	79	83	89	97



- This app shows how to use the Java Executor framework to check if N random #'s are prime
  - Each natural # divisible only by 1 & itself is prime

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 333102181 is not prime with smallest factor 3 030676473 is not prime with smallest factor 619 27457798 is not prime with smallest factor 2 583326869 is prime 16682593 is not prime with smallest factor 11 509282196 is not prime with smallest factor 2 55195772 is not prime with smallest factor 2 320523007 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 699167856 is not prime with smallest factor 2 1563413821 is prime Finished primality computations The user can select the # 'N'

UE 9 4:37

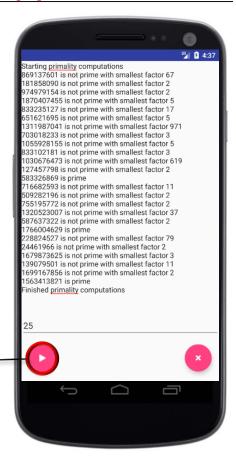
Starting primality computations

869137601 is not prime with smallest factor 67 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

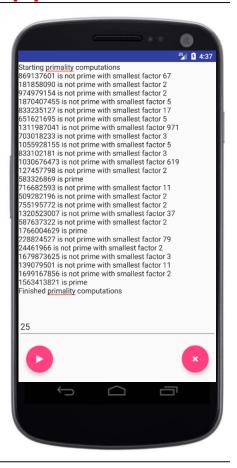
- This app shows how to use the Java Executor framework to check if N random #'s are prime
  - Each natural # divisible only by 1 & itself is prime

The user can also start running the app



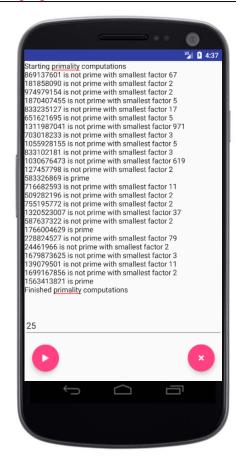
This app has several notable properties



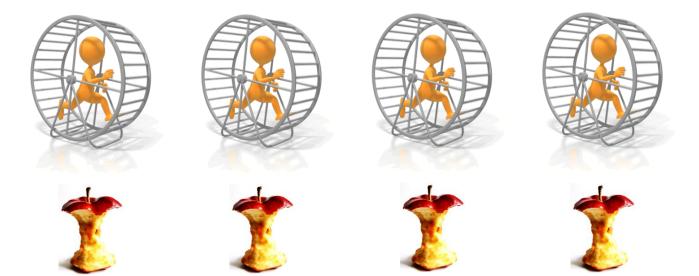


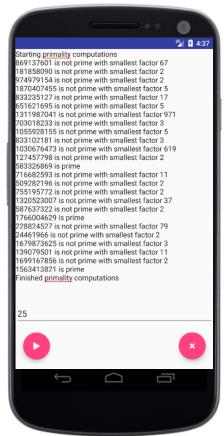
- This app has several notable properties
  - It is "embarrassingly parallel"
    - i.e., no data dependencies between worker threads





- This app has several notable properties
  - It is "embarrassingly parallel"
  - It is compute-bound
    - i.e., time to complete a task is dictated by CPU speed





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

 PrimeRunnable defines a brute-force means to check if <<Java Class>> • MainActivity a # is prime MainActivity() long isPrime(long n) { onCreate(Bundle):void ■ initializeViews(Bundle):void if (n > 3)setCount(View):void for (long factor = 2; handleStartButton(View):void factor <= n / 2;</pre> startComputations(int):void odone():void ++factor) println(String):void if (n / factor \* factor onResume():void == n)-mActivity 0..1 return factor; <<Java Class>> return 0; PrimeRunnable PrimeRunnable(MainActivity,long) ■ isPrime(long,long,long):long run():void

See www.mkyong.com/java/how-to-determine-a-prime-number-in-java

is "compute-bound"

 PrimeRunnable defines a brute-force means to check if a # is prime

```
long isPrime(long n) {
  if (n > 3)
    for (long factor = 2;
         factor <= n / 2;
         ++factor)
       if (n / factor * factor
           == n)
         return factor;
  return 0;
                     Note how this algorithm
```

• MainActivity MainActivity() onCreate(Bundle):void ■ initializeViews(Bundle):void setCount(View):void handleStartButton(View):void startComputations(int):void odone():void println(String):void onResume():void -mActivity 0..1 <<Java Class>> PrimeRunnable

PrimeRunnable(MainActivity,long)

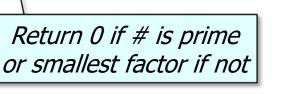
■ isPrime(long,long,long):long

run():void

<<Java Class>>

 PrimeRunnable defines a brute-force means to check if a # is prime

```
long isPrime(long n) {
  if (n > 3)
    for (long factor = 2;
         factor <= n / 2;
         ++factor)
       if (n / factor * factor
           == n)
         return factor;
  return 0;
```





• MainActivity MainActivity() onCreate(Bundle):void ■ initializeViews(Bundle):void setCount(View):void handleStartButton(View):void startComputations(int):void done():void println(String):void onResume():void -mActivity 0..1 <<Java Class>> PrimeRunnable

PrimeRunnable(MainActivity,long)

■ isPrime(long,long,long):long

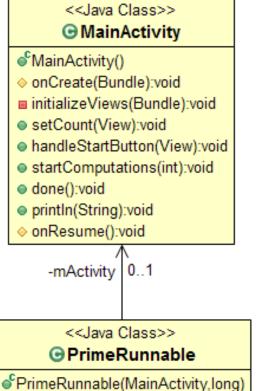
run():void

<<Java Class>>

The goal is to burn non-trivial CPU time!!

 This app uses a Java Executor that's implemented w/a fixed-size thread pool tuned to the # of processor cores in the computing device

Creates a thread pool that reuses a fixed # of threads operating off a shared unbounded queue



isPrime(long,long,long):long

run():void

 This app uses a Java Executor that's implemented w/a fixed-size thread pool tuned to the # of processor cores in the computing device

• MainActivity MainActivity() onCreate(Bundle):void ■ initializeViews(Bundle):void setCount(View):void handleStartButton(View):void startComputations(int):void done():void println(String):void onResume():void -mActivity 0..1 <<Java Class>> PrimeRunnable PrimeRunnable(MainActivity,long)

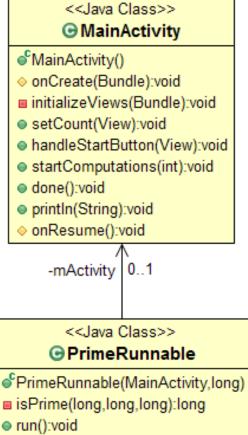
isPrime(long,long,long):long

run():void

<<Java Class>>

 This app uses a Java Executor that's implemented w/a fixed-size thread pool tuned to the # of processor cores in the computing device

is inherently a "compute-bound" task



 MainActivity creates/executes a PrimeRunnable for each of the "count" random # new Random()

```
.forEach(randomNumber ->
        mExecutor.execute
```

.longs(count,

(new PrimeRunnable

```
(this, randomNumber)));
```

sMAX VALUE - count, sMAX VALUE)

<<Java Class>> • MainActivity MainActivity() onCreate(Bundle):void ■ initializeViews(Bundle):void setCount(View):void handleStartButton(View):void startComputations(int):void done():void println(String):void onResume():void -mActivity <<Java Class>>

PrimeRunnable

- PrimeRunnable(MainActivity,long) isPrime(long,long,long):long
- run():void

```
of the "count" random #

These random longs are in the range sMAX_VALUE - count & sMAX_VALUE

sMAX_VALUE - count, sMAX_VALUE)
```

setCount(View):void handleStartButton(View):void startComputations(int):void done():void println(String):void onResume():void -mActivity 0..1 <<Java Class>> PrimeRunnable PrimeRunnable(MainActivity,long)

isPrime(long,long,long):long

run():void

onCreate(Bundle):void

■ initializeViews(Bundle):void

sMAX\_VALUE is set to a large #, e.g., 1,000,000,000

```
• MainActivity
of the "count" random #
                                                                      MainActivity()
                                                                      onCreate(Bundle):void
                                                                      initializeViews(Bundle):void
new Random()
                                                                      setCount(View):void
    .longs(count,
                                                                      handleStartButton(View):void
                                                                      startComputations(int):void
              sMAX VALUE - count, sMAX VALUE)
                                                                      done():void
                                                                      println(String):void
    .forEach(randomNumber ->
                                                                      onResume():void
                mExecutor.execute
                                                                         -mActivity 0..1
                    (new PrimeRunnable
                              (this, randomNumber)));
                                                                           <<Java Class>>
                                                                         PrimeRunnable
                       Each random long is gueued for
                                                                     PrimeRunnable(MainActivity,long)
                       execution by a thread in the pool
                                                                     isPrime(long,long,long):long
                                                                     run():void
```

See <a href="mailto:docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html#execute">docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html#execute</a>

MainActivity creates/executes a PrimeRunnable for each

```
<<Java Class>>
                                                                            • MainActivity
of the "count" random #
                                                                       MainActivity()
                                                                       onCreate(Bundle):void
                                                                       ■ initializeViews(Bundle):void
new Random()
                                                                       setCount(View):void
    .longs(count,
                                                                       handleStartButton(View):void
                                                                       startComputations(int):void
              sMAX VALUE - count, sMAX VALUE)
                                                                       odone():void
                                                                       println(String):void
    .forEach(randomNumber ->
                                                                       onResume():void
                mExecutor.execute
                                                                          -mActivity 0..1
                    (new PrimeRunnable
                               (this, randomNumber)));
                                                                            <<Java Class>>
                                                                          PrimeRunnable
                        Each random long is queued for
                                                                     PrimeRunnable(MainActivity,long)
                       execution by a thread in the pool
                                                                     isPrime(long,long,long):long
                                                                     run():void
```

See <a href="mailto:docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html#execute">docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html#execute</a>

PrimeRunnable determines if a # is prime

class PrimeRunnable implements Runnable {

long mPrimeCandidate;

private final MainActivity mActivity;

..

•

PrimeRunnable(MainActivity a, Long pc)
{ mActivity = a; mPrimeCandidate = pc; }

long isPrime(long n) { ... }

PrimeResult run() {
 long smallestFactor =

isPrime (mPrimeCandidate);

MainActivity()
 onCreate(Bundle):void
 initializeViews(Bundle):void
 setCount(View):void
 handleStartButton(View):void

<<Java Class>>

MainActivity

startComputations(int):void
 done():void
 println(String):void

onResume():void

↑
-mActivity 0..1

<<Java Class>>

<<Java Class>>

PrimeRunnable

© PrimeRunnable(MainActivity,long)
■ isPrime(long,long,long):long
● run():void

See PrimeExecutor/app/src/main/java/vandy/mooc/prime/activities/PrimeRunnable.java

Overview of the PrimeChecker App PrimeRunnable determines if a # is prime <<Java Class>> • MainActivity class PrimeRunnable implements Runnable { long mPrimeCandidate; onCreate(Bundle):void private final MainActivity mActivity; ■ initializeViews(Bundle):void setCount(View):void Implements Runnable handleStartButton(View):void startComputations(int):void PrimeRunnable (MainActivity a, Long pc) odone():void println(String):void { mActivity = a; mPrimeCandidate = pc; } onResume():void

long isPrime(long n) { ... }

PrimeResult run() {
 long smallestFactor =
 isPrime(mPrimeCandidate);
 } ...

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

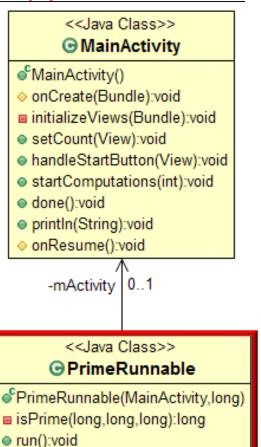
 PrimeRunnable determines if a # is prime class PrimeRunnable implements Runnable { long mPrimeCandidate; private final MainActivity mActivity; Constructor stores prime # candidate & activity PrimeRunnable (MainActivity a, Long pc) { mActivity = a; mPrimeCandidate = pc; } long isPrime(long n) { ... } PrimeResult run() { long smallestFactor = isPrime (mPrimeCandidate);

<<Java Class>> • MainActivity onCreate(Bundle):void ■ initializeViews(Bundle):void setCount(View):void handleStartButton(View):void startComputations(int):void odone():void println(String):void onResume():void -mActivity <<Java Class>> PrimeRunnable PrimeRunnable(MainActivity,long)

isPrime(long,long,long):long

run():void

 PrimeRunnable determines if a # is prime class PrimeRunnable implements Runnable { long mPrimeCandidate; private final MainActivity mActivity; PrimeRunnable (MainActivity a, Long pc) { mActivity = a; mPrimeCandidate = pc; } Returns 0 if n is prime or long isPrime(long n) smallest factor if it's not PrimeResult run() { long smallestFactor = isPrime (mPrimeCandidate);

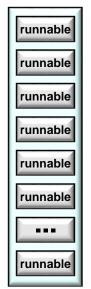


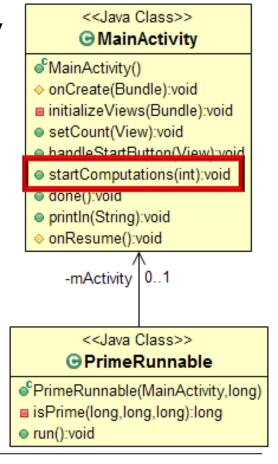
 PrimeRunnable determines if a # is prime class PrimeRunnable implements Runnable { long mPrimeCandidate; private final MainActivity mActivity; PrimeRunnable (MainActivity a, Long pc) { mActivity = a; mPrimeCandidate = pc; } long isPrime(long n) { ... } PrimeResult run() { long smallestFactor = isPrime (mPrimeCandidate);

```
<<Java Class>>
       • MainActivity
  onCreate(Bundle):void
  ■ initializeViews(Bundle):void
  setCount(View):void
  handleStartButton(View):void
  startComputations(int):void
  odone():void
  println(String):void
 onResume():void
     -mActivity 0..1
        <<Java Class>>
     PrimeRunnable
PrimeRunnable(MainActivity,long)
isPrime(long,long,long):long
run():void
```

 Although there may be many PrimeRunnable instances, they will run on a (much) smaller # of threads, which can be tuned transparently

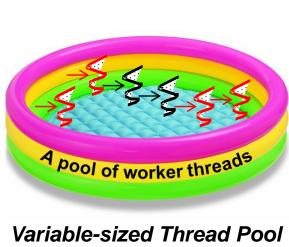


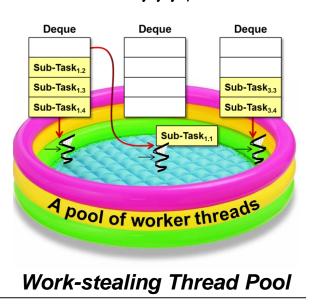




 The Java Executor interface enables the # & type of threads to be tuned transparently wrt the prime checker app logic

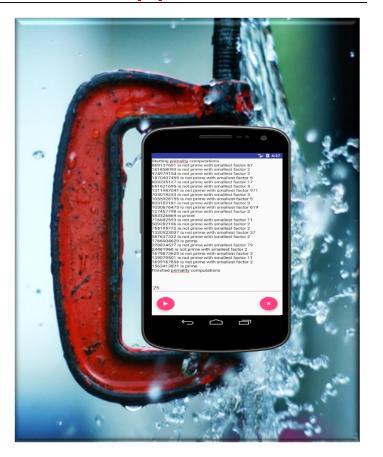






28

However, Java Executor has some restrictions



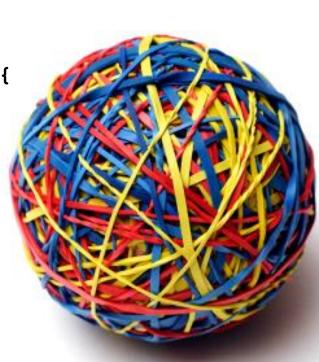
- However, Java Executor has some restrictions, e.g.
  - One-way semantics of runnables tightly couple PrimeRunnable with MainActivity



```
class PrimeRunnable implements Runnable {
 private final MainActivity mActivity;
  public PrimeRunnable(MainActivity activity)
  { mActivity = activity; ... }
 public void run() {
    ... mActivity.done(); ...
```

This tight coupling complicates runtime configuration changes

- However, Java Executor has some restrictions, e.g.
  - One-way semantics of runnables tightly couple PrimeRunnable with MainActivity
  - isPrime() tightly coupled w/PrimeRunnable class PrimeRunnable implements Runnable { long isPrime(long n) { if (n > 3)for (long factor = 2; factor <= n / 2; ++factor)</pre> if (n / factor \* factor == n) return factor; return 0;



However, Java Executor has some restrictions, e.g.

 One-way semantics of runnables tightly couple PrimeRunnable with MainActivity

• isPrime() tightly coupled w/PrimeRunnable

 The lack of lifecycle operations on Java Executor



- However, Java Executor has some restrictions, e.g.
  - One-way semantics of runnables tightly couple PrimeRunnable with MainActivity
  - isPrime() tightly coupled w/PrimeRunnable
  - The lack of lifecycle operations on Java Executor, e.g.
    - Can't interrupt/cancel running tasks



- However, Java Executor has some restrictions, e.g.
  - One-way semantics of runnables tightly couple PrimeRunnable with MainActivity
  - isPrime() tightly coupled w/PrimeRunnable
  - The lack of lifecycle operations on Java Executor, e.g.
    - Can't interrupt/cancel running tasks
    - Can't handle runtime configuration changes gracefully
      - e.g., must restart processing from the beginning



# End of Overview of Java Executor Interface (Part 2)