Java Executor:

Application to the 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

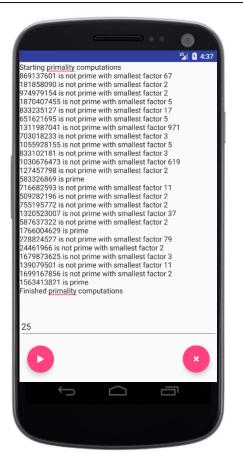
Vanderbilt University Nashville, Tennessee, USA



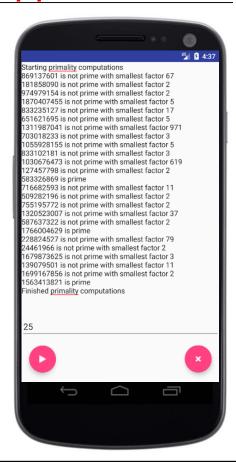
Learning Objectives in this Part of the Lesson

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



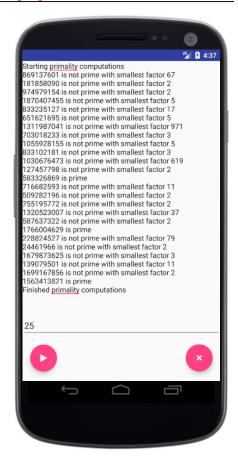


 This app shows how to use the Java Executor framework to check if N random #'s are prime



- 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

509282196 is not prime with smallest factor 2
755195772 is not prime with smallest factor 2
1320523007 is not prime with smallest factor 3
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 3
139079501 is not prime with smallest factor 2
1563413821 is prime
Finished primality computations

The user can select the # 'N'

Starting primality computations

583326869 is prime

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 651621695 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 3 1030676473 is not prime with smallest factor 619 127457798 is not prime with smallest factor 619

16682593 is not prime with smallest factor 11

- 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 also start running the app

UE 9 4:37

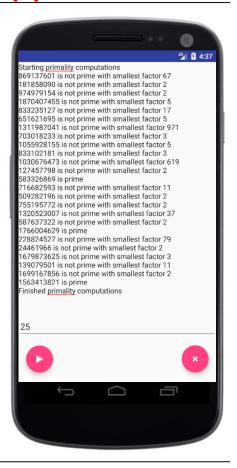
Starting primality computations

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

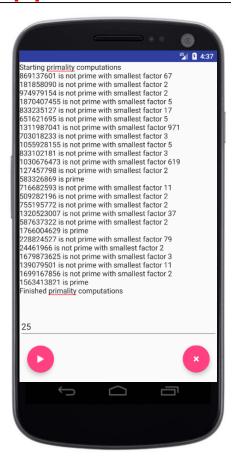
This app has several notable properties



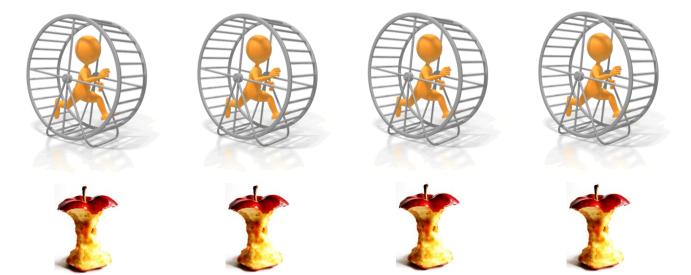


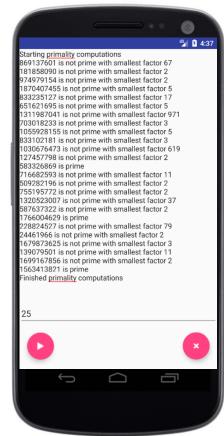
- This app has several notable properties
 - It is "embarrassingly parallel"
 - i.e., no data dependencies between running tasks





- 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

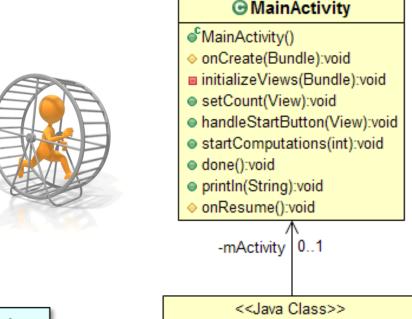
 PrimeRunnable defines a brute-force means to check if <<Java Class>> • MainActivity a # is prime MainActivity() long isPrime(long n) { onCreate(Bundle):void ■ initializeViews(Bundle):void if (n > 3)setCount(View):void for (long factor = 2; handleStartButton(View):void factor <= n / 2; startComputations(int):void 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

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

return 0;

Note how this algorithm is "compute-bound"



run():void

<<Java Class>>

PrimeRunnable

PrimeRunnable(MainActivity,long)

■ isPrime(long,long,long):long

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

Return 0 if # is prime

or smallest factor if not

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



<<Java Class>> • 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

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

78134 63873

• This app uses a fixed-sized Executor implementation 73735 45963

```
02965 58303
                                                                               90708 20025
                                                                        23851
                                                                               27965 62394
                                                                   33666 62570
                                                                               64775 78428
                                                                   81666 26440
                                                                               20422 05720
mExecutor = Executors
   .newFixedThreadPool
                                                                  Stream of Random Numbers
        (Runtime.getRuntime().
                                                                                   run()
                                                              execute()
         availableProcessors());
                                                                2.offer()
 Create fixed-sized
                                                                              Fixed
                                                                   runnable
    thread pool
                                                                              WorkerThreads
                                                    runnable
                                                                   runnable
                                   UI Thread
                                                  1.execute
                                                                   runnable
                                   (main thread)
                                                     (task)
                                                                                3.take()
                                                                   runnable
                                                                               4.run()
                                                                 WorkQueue
                                                                                 runnable
                                                                    ThreadPoolExecutor
```

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

78134 63873

• This app uses a fixed-sized Executor implementation 73735 45963

```
23851
                                                                            27965 62394
                                                                 33666 62570
                                                                 81666 26440
mExecutor = Executors
   .newFixedThreadPool
                                                                Stream of Random Numbers
       (Runtime.getRuntime().
                                                                                run()
                                                            execute()
        availableProcessors());
                                                              2.offer()
Returns # of processor cores known
                                                                           Fixed
                                                                runnable
 to the Java execution environment
                                                                           WorkerThreads
                                                  runnable
                                                                runnable
                                  UI Thread
                                                 1.execute
                                                                runnable
                                  (main thread)
                                                   (task)
                                                                             3. take()
                                                                runnable
                                                                             4.run()
                                                               WorkQueue
                                                                              runnable
                                                                  ThreadPoolExecutor
```

73735 45963

78134 63873

This app uses a fixed-sized Executor implementation

```
02965 58303
                                                                               90708 20025
                                                                   98859 23851
                                                                               27965 62394
                                                                   33666 62570
                                                                               64775 78428
                                                                   81666 26440
                                                                               20422 05720
mExecutor = Executors
   .newFixedThreadPool
                                                                  Stream of Random Numbers
        (Runtime.getRuntime().
                                                                                   run()
                                                              execute()
         availableProcessors());
                                                                2.offer()
 Use this value since isPrime()
  is a "compute-bound" task
                                                                             Fixed
                                                                  runnable
                                                                             WorkerThreads
                                                    runnable
                                                                  runnable
                                   UI Thread
                                                  1.execute
                                                                  runnable
                                   (main thread)
                                                     (task)
                                                                               3. take()
                                                                  runnable
                                                                               4.run()
                                                                 WorkQueue
                                                                                runnable
                                                                    ThreadPoolExecutor
```

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

 This app uses a fixed-sized Executor implementation 73735 45963 78134 63873 02965 58303 90708 20025 98859 23851 27965 62394 33666 62570 64775 78428 81666 26440 20422 05720 mExecutor = Executors .newFixedThreadPool Stream of Random Numbers (Runtime.getRuntime(). run() execute() availableProcessors()); 2.offer() **Fixed** runnable WorkerThreads runnable runnable **UI Thread**

(main thread)

The UI thread generates random #'s that are processed via the thread pool

 This app uses a fixed-sized Executor implementation 73735 45963 78134 63873 90708 20025 02965 58303 98859 23851 27965 62394 33666 62570 64775 78428 81666 26440 20422 05720 mExecutor = Executors .newFixedThreadPool Stream of Random Numbers (Runtime.getRuntime(). run() execute() availableProcessors()); 2.offer() Fixed runnable WorkerThreads runnable runnable **UI Thread** 1.execute runnable (main thread) (task) 3. take() runnable 4.run() This fixed-size thread pool uses an WorkQueue runnable unbounded queue to avoid deadlocks **ThreadPoolExecutor**

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

 MainActivity creates/executes a PrimeRunnable for each <<Java Class>> • MainActivity of the "count" random # new Random() .longs(count, sMAX VALUE - count, sMAX VALUE) done():void .forEach(randomNumber -> mExecutor.execute

(new PrimeRunnable

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

run():void

(this, randomNumber)));

 MainActivity creates/executes a PrimeRunnable for each of the "count" random #

```
These random longs are in the range
new Random()
                  SMAX_VALUE - count & SMAX_VALUE
  .longs(count,
         sMAX VALUE - count, sMAX VALUE)
  .forEach(randomNumber ->
            mExecutor.execute
              (new PrimeRunnable
                      (this, randomNumber)));
```

```
<<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
```

 MainActivity creates/executes a PrimeRunnable for each of the "count" random #

new Random()
.longs(count,

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

sMAX_VALUE - count, sMAX_VALUE)

.forEach(randomNumber ->



mExecutor.execute
(new PrimeRunnable

(this, randomNumber)));

• 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>>

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 docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html#execute

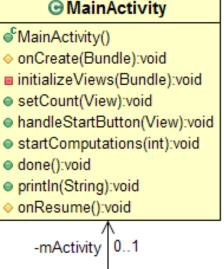
```
class PrimeRunnable implements Runnable {
  long mPrimeCandidate;
  private final MainActivity mActivity;
```

```
PrimeRunnable (MainActivity a, Long pc)
```

{ mActivity = a; mPrimeCandidate = pc; }

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

```
void run() {
  long smallestFactor =
    isPrime(mPrimeCandidate);
```





© 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 -mActivity 0..1 long isPrime(long n) { ... }

-mActivity 0..1

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

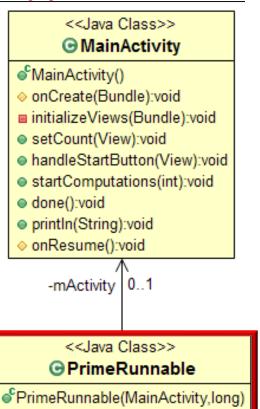
• MainActivity onCreate(Bundle):void ■ initializeViews(Bundle):void setCount(View):void handleStartButton(View):void startComputations(int):void odone():void println(String):void onResume():void -mActivity <<Java Class>> PrimeRunnable PrimeRunnable(MainActivity,long)

isPrime(long,long,long):long

run():void

<<Java Class>>

 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 void run() long smallestFactor = isPrime (mPrimeCandidate);



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; } long isPrime(long n) { ... } void run() { long smallestFactor = isPrime (mPrimeCandidate); run() hook method invokes isPrime() in a pool thread

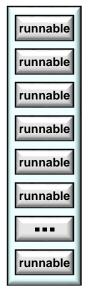
<<Java Class>> • MainActivity onCreate(Bundle):void ■ initializeViews(Bundle):void setCount(View):void handleStartButton(View):void startComputations(int):void odone():void println(String):void onResume():void -mActivity 0..1 <<Java Class>> PrimeRunnable PrimeRunnable(MainActivity,long)

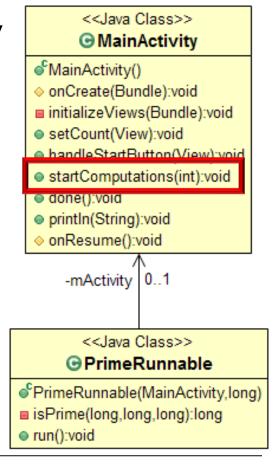
■ isPrime(long,long,long):long

run():void

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







End of Java Executor: Application to PrimeChecker App