The Java Executor Interface (Part 2)

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

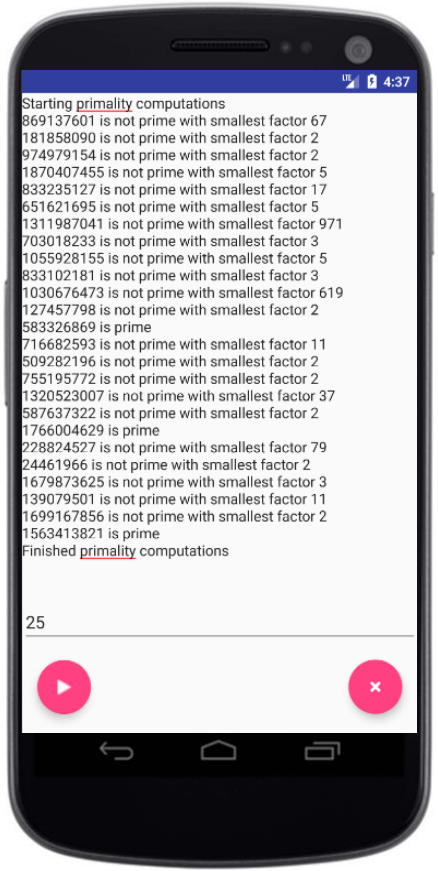
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

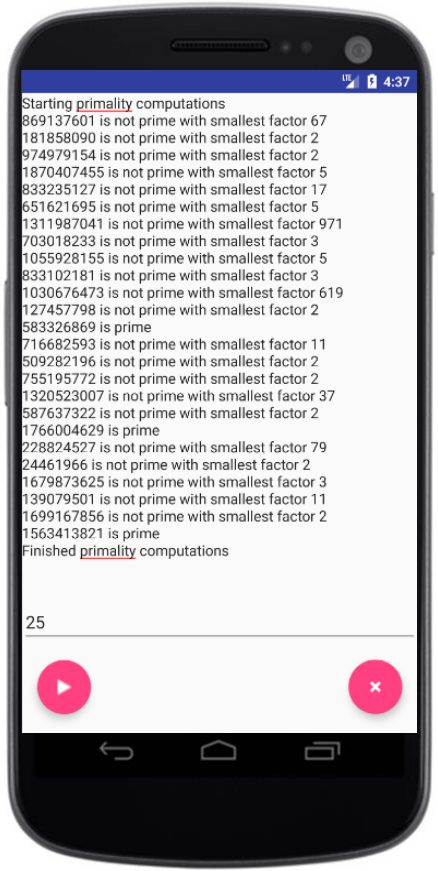
- Recognize the simple/single feature provided by the Java Executor interface
- Learn how to program a simple "prime checker" app using the Java Executor interface



Overview of the PrimeChecker App

Overview of the PrimeChecker App

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

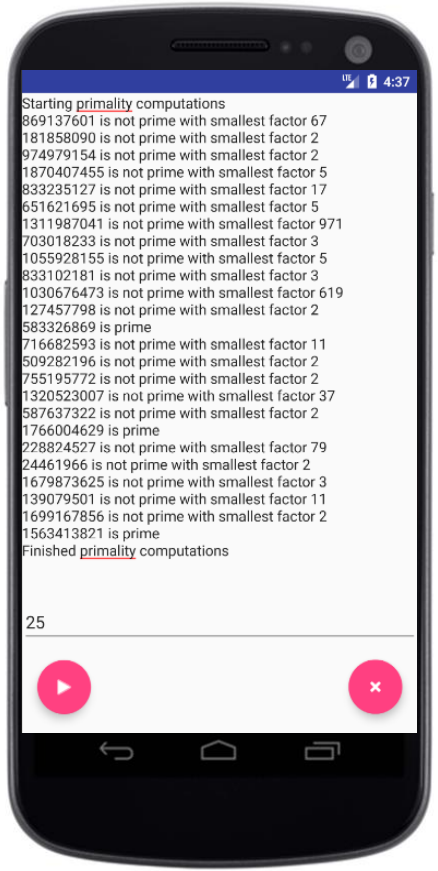


See github.com/douglascraigschmidt/POSA/tree/master/ex/M4/Primes/PrimeExecutor

Overview of the PrimeChecker App

- 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

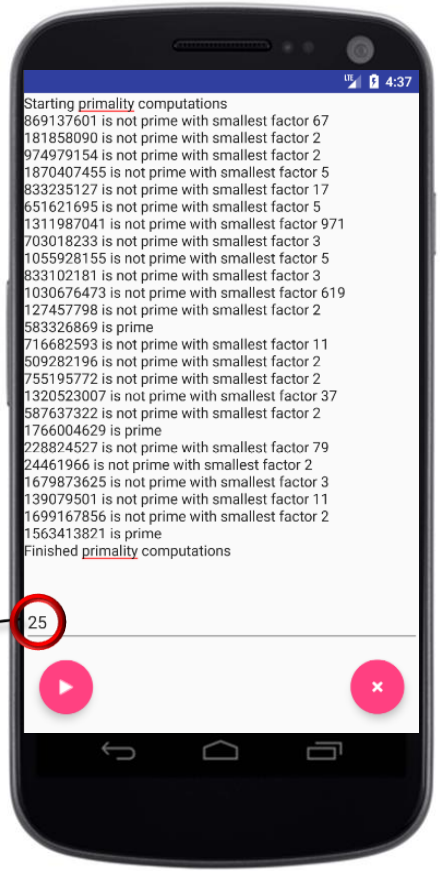
2	3	5	7	11
13	17	19	23	29
31	37	41	43	47
53	59	61	67	71
73	79	83	89	97



See en.wikipedia.org/wiki/Prime_number

Overview of the PrimeChecker App

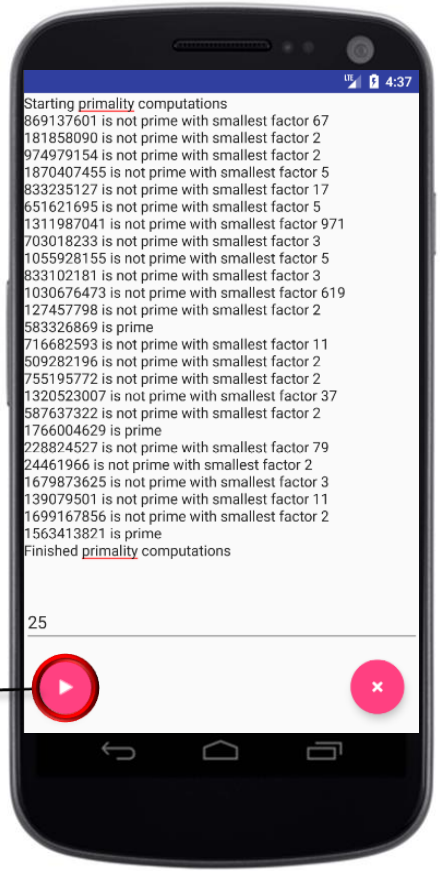
- This app shows how to use the Java Executor framework to check if *N* random #'s are prime
  - Each natural # divisible only by 1 & itself is prime



The user can select the # 'N'

Overview of the PrimeChecker App

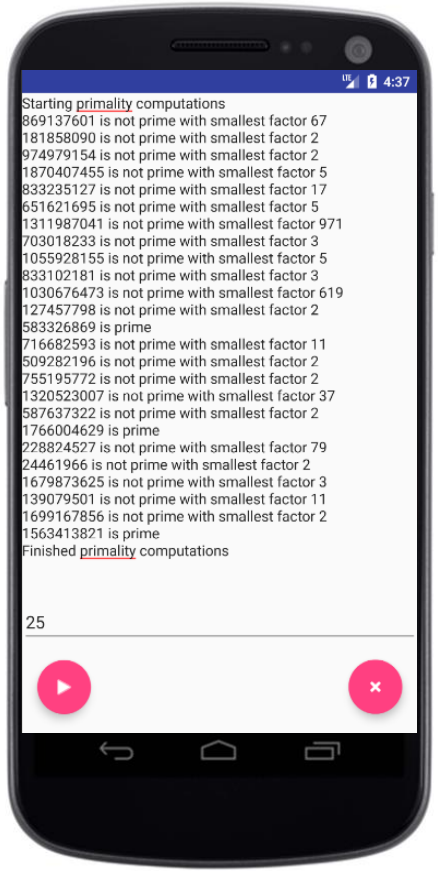
- This app shows how to use the Java Executor framework to check if *N* random #'s are prime
  - Each natural # divisible only by 1 & itself is prime



The user can also start running the app

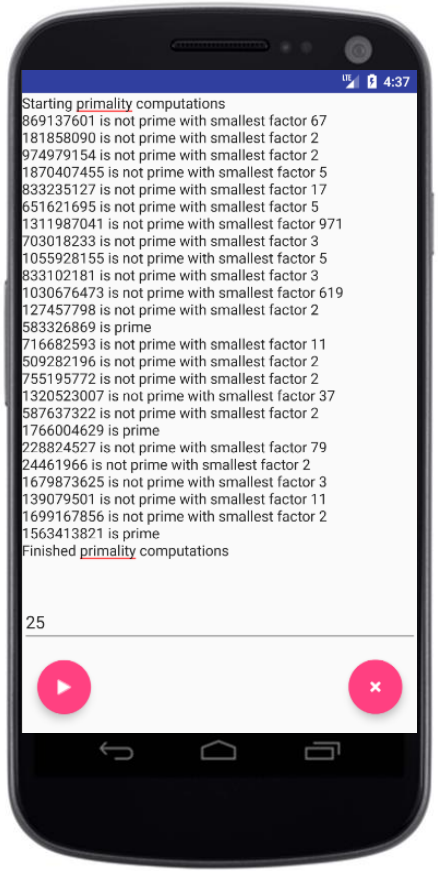
Overview of the PrimeChecker App

- This app has several notable properties



Overview of the PrimeChecker App

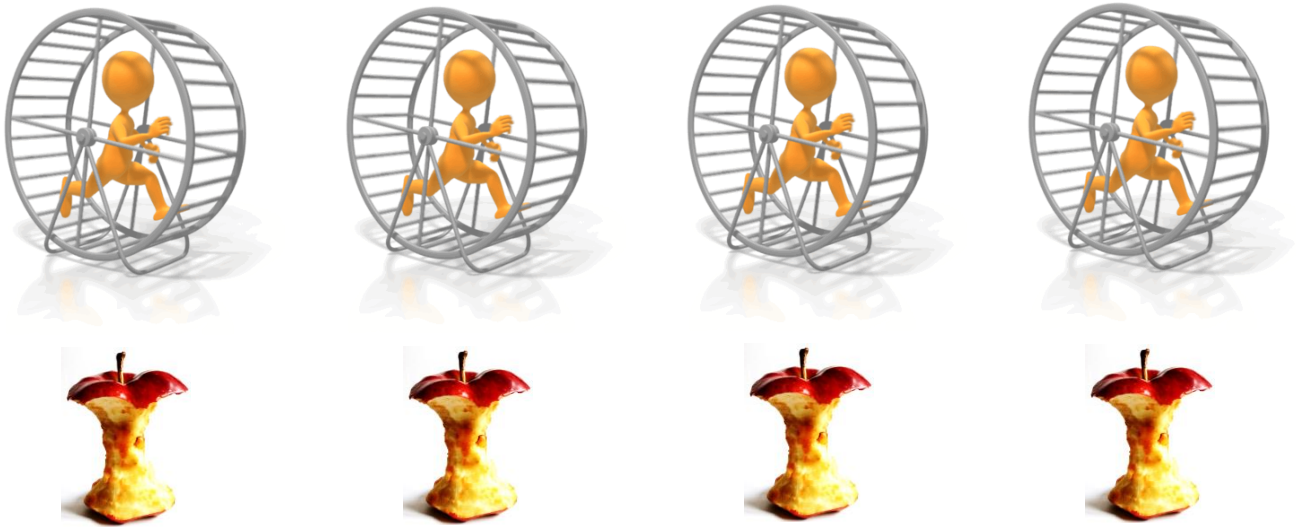
- This app has several notable properties
  - It is "embarrassingly parallel"
    - *i.e., no data dependencies between worker threads*

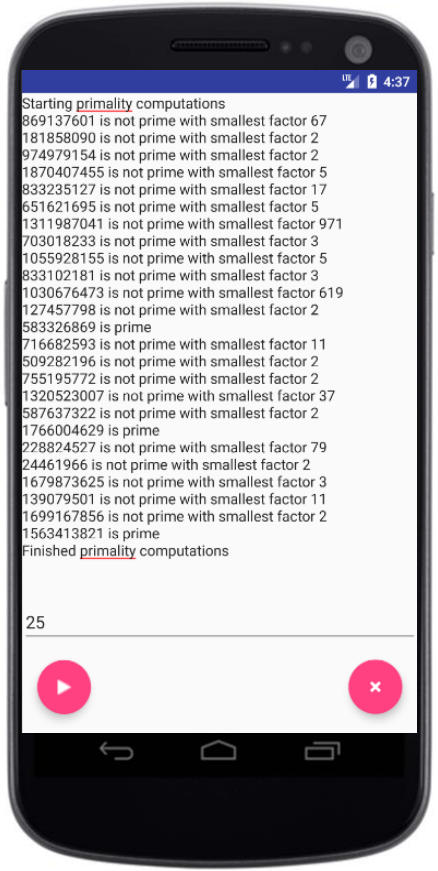


See en.wikipedia.org/wiki/Embarrassingly_parallel

Overview of the PrimeChecker App

- 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



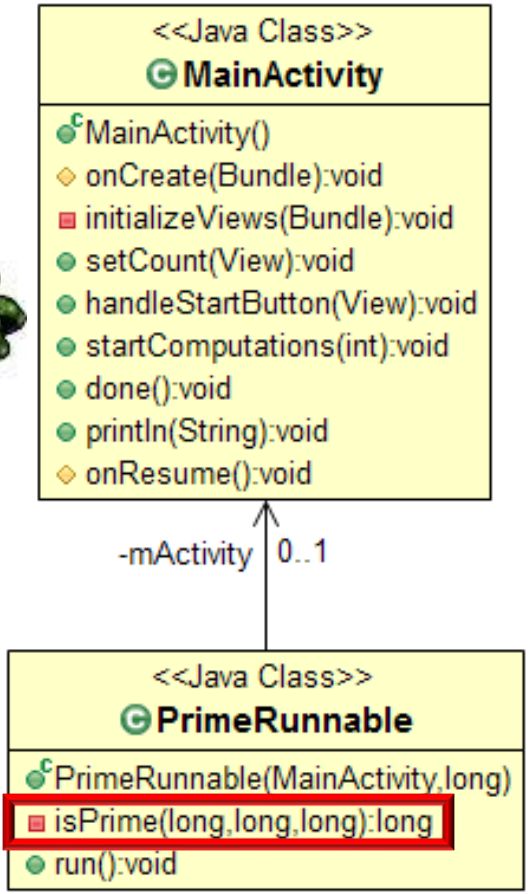


See en.wikipedia.org/wiki/CPU-bound

Overview of the PrimeChecker App

- 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;
}
```



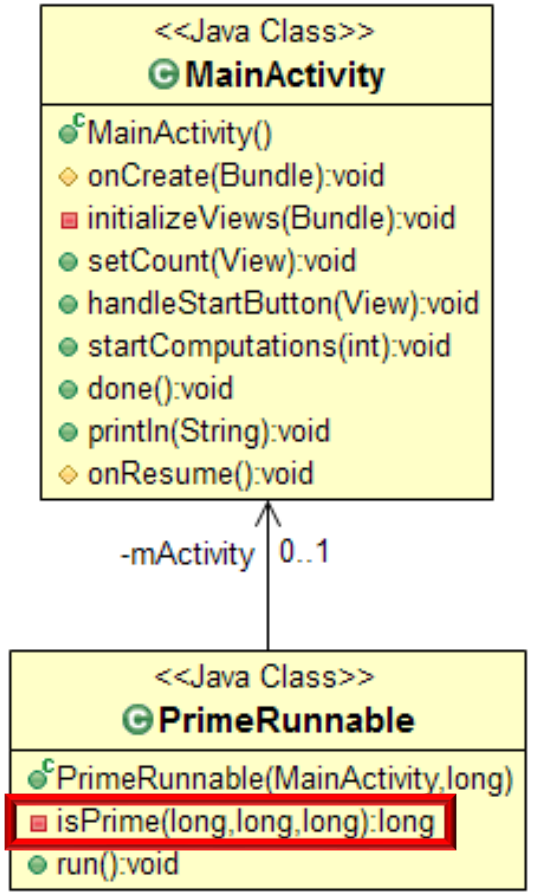
See www.mkyong.com/java/how-to-determine-a-prime-number-in-java

Overview of the PrimeChecker App

- 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 is "compute-bound"



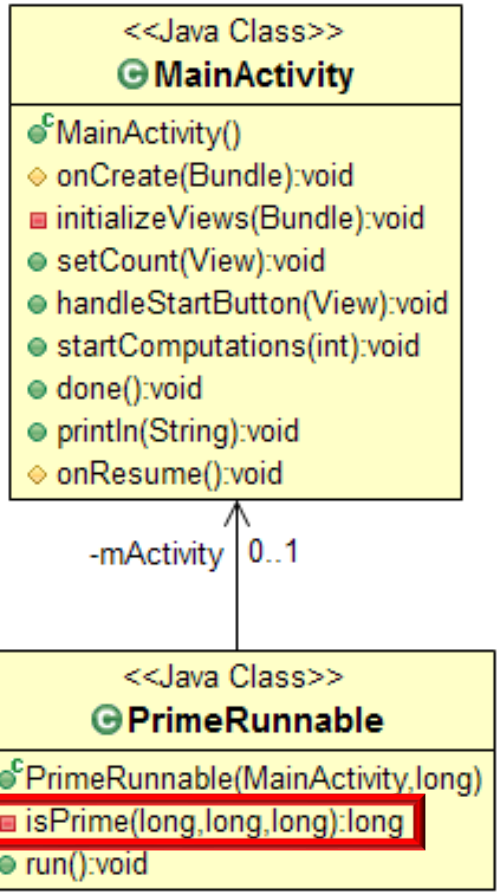
See en.wikipedia.org/wiki/CPU-bound

Overview of the PrimeChecker App

- 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;
}
```

Return 0 if # is prime or smallest factor if not



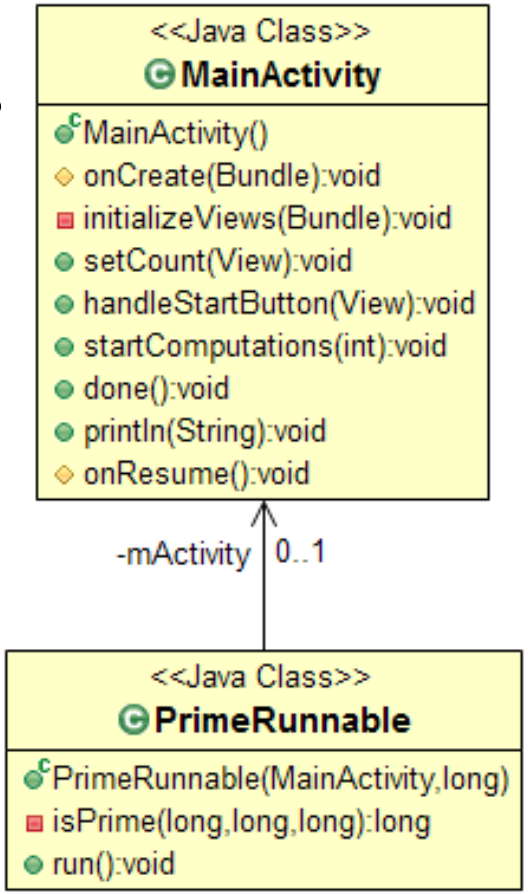
The goal is to burn non-trivial CPU time!!

Overview of the PrimeChecker App

- 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

```
mExecutor = Executors.newFixedThreadPool
              (Runtime.getRuntime().
               availableProcessors());
```

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



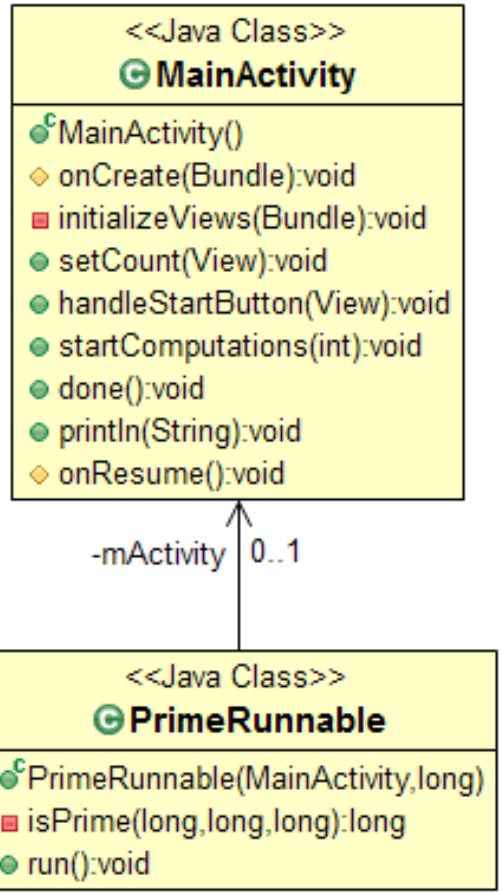
See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newFixedThreadPool

Overview of the PrimeChecker App

- 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

```
mExecutor = Executors.newFixedThreadPool
                (Runtime.getRuntime().
                 availableProcessors());
```

Returns # of processors available to the Java execution environment



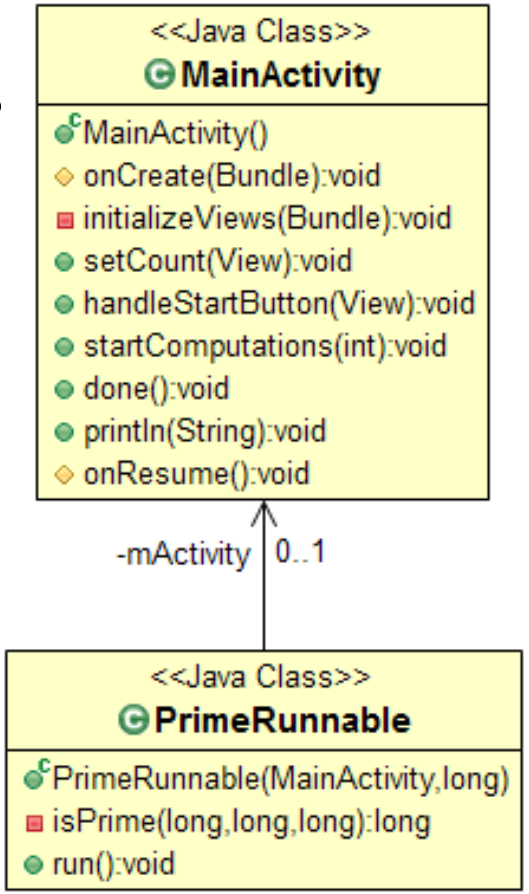
See docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#availableProcessors

Overview of the PrimeChecker App

- 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

```
mExecutor = Executors.newFixedThreadPool
                (Runtime.getRuntime().
                 availableProcessors());
```

This value is suitable since isPrime() is inherently a "compute-bound" task

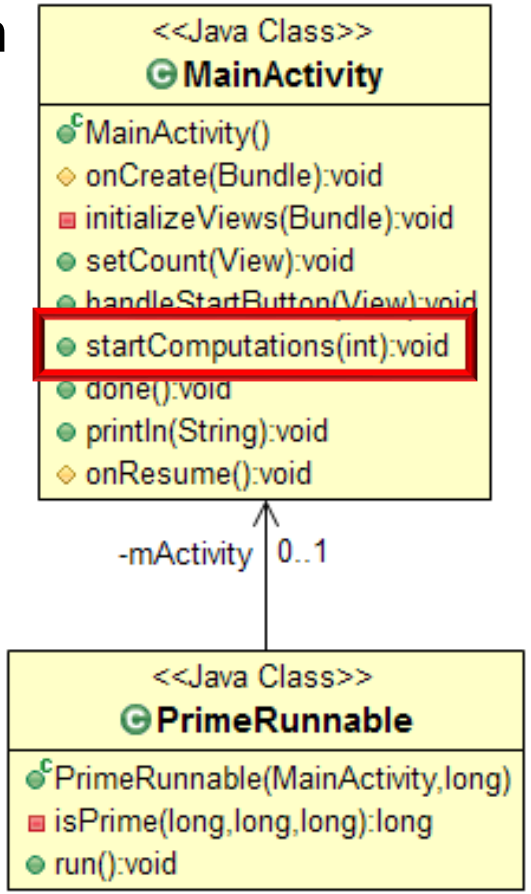


See en.wikipedia.org/wiki/CPU-bound

Overview of the PrimeChecker App

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

```
new Random()
   .longs(count,
          sMAX_VALUE - count, sMAX_VALUE)

   .forEach(randomNumber ->
            mExecutor.execute
               (new PrimeRunnable
                     (this, randomNumber)));
```

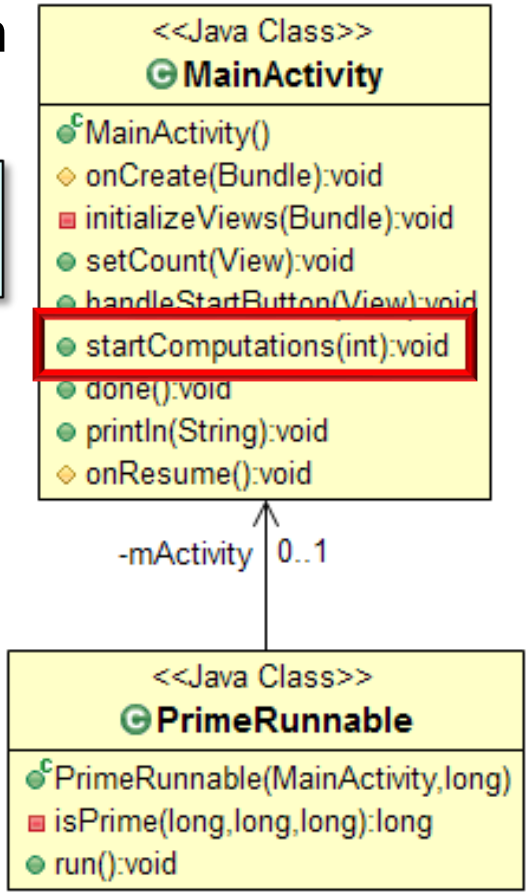


Overview of the PrimeChecker App

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

```
new Random()
   .longs(count,
          sMAX_VALUE - count, sMAX_VALUE)

   .forEach(randomNumber ->
            mExecutor.execute
               (new PrimeRunnable
                     (this, randomNumber)));
```

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



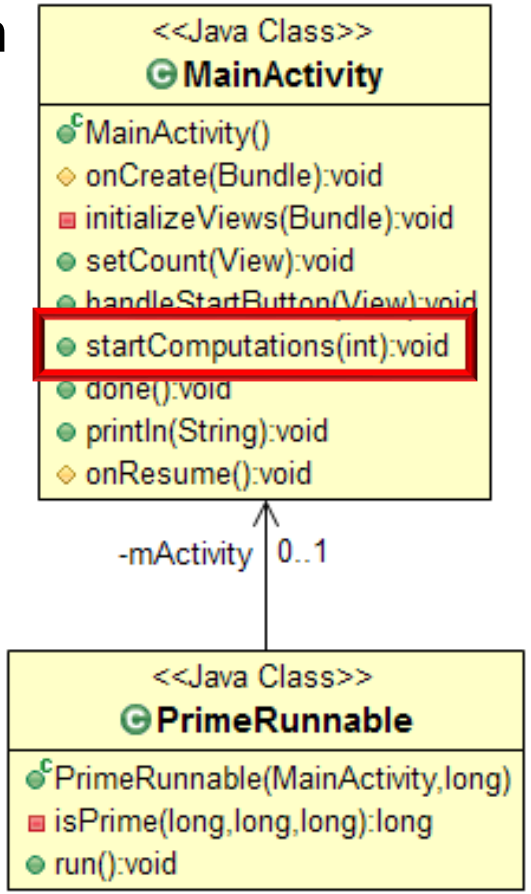
sMAX_VALUE is set to a large #, e.g., 1,000,000,000

Overview of the PrimeChecker App

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

```
new Random()
   .longs(count,
          sMAX_VALUE - count, sMAX_VALUE)

   .forEach(randomNumber ->
            mExecutor.execute
               (new PrimeRunnable
                     (this, randomNumber)));
```

Each random long is queued for execution by a thread in the pool



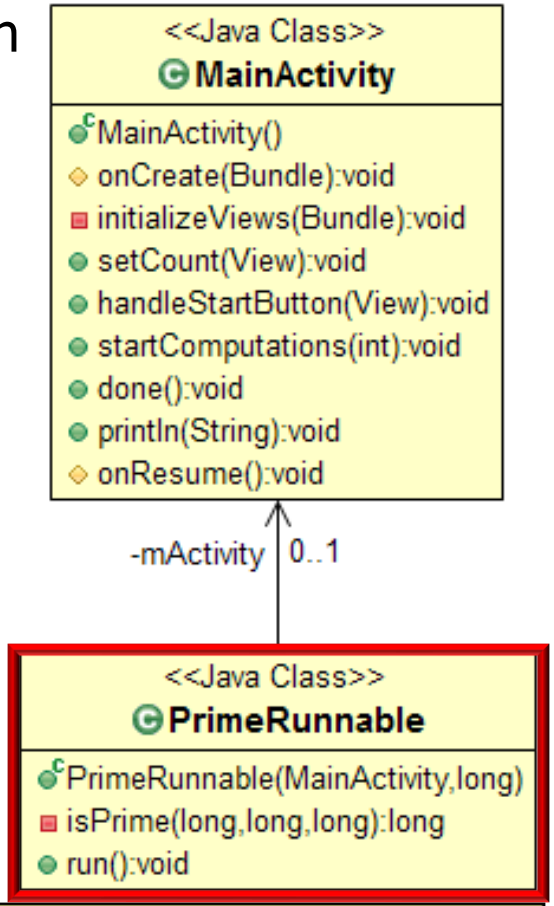
See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html#execute

Overview of the PrimeChecker App

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

```
new Random()
   .longs(count,
          sMAX_VALUE - count, sMAX_VALUE)

   .forEach(randomNumber ->
            mExecutor.execute
               (new PrimeRunnable
                     (this, randomNumber)));
```

Each random long is queued for execution by a thread in the pool

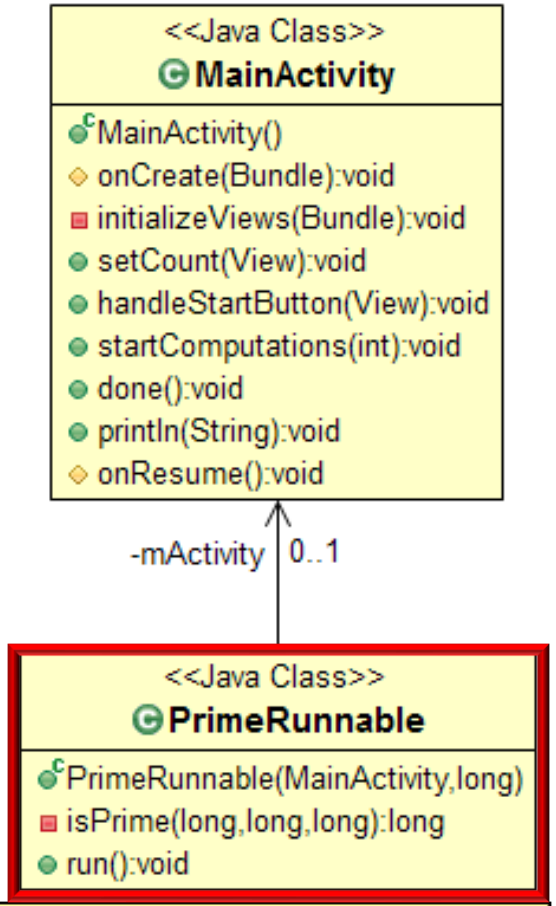


See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html#execute

Overview of the PrimeChecker App

- 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);
  } ...
```



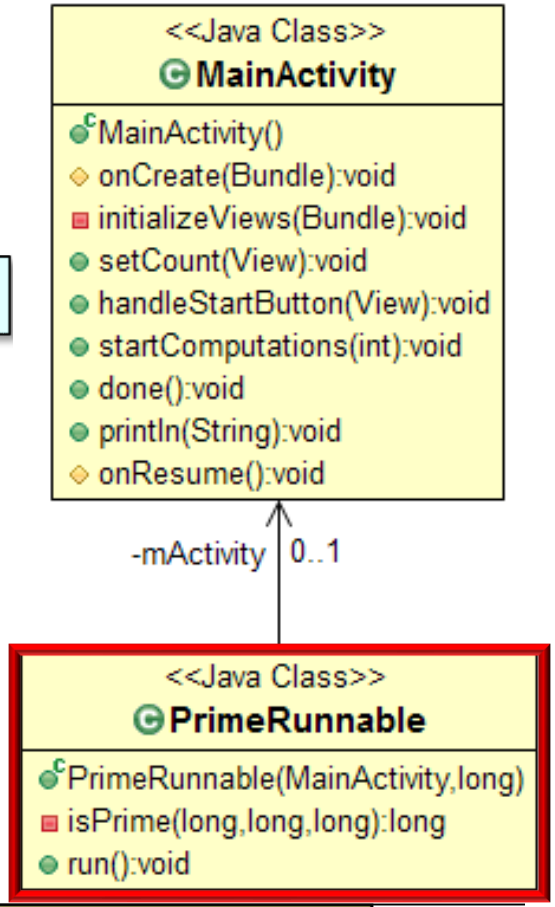
See PrimeExecutor/app/src/main/java/vandy/mooc/prime/activities/PrimeRunnable.java

Overview of the PrimeChecker App

- 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);
  } ...
```

Implements Runnable



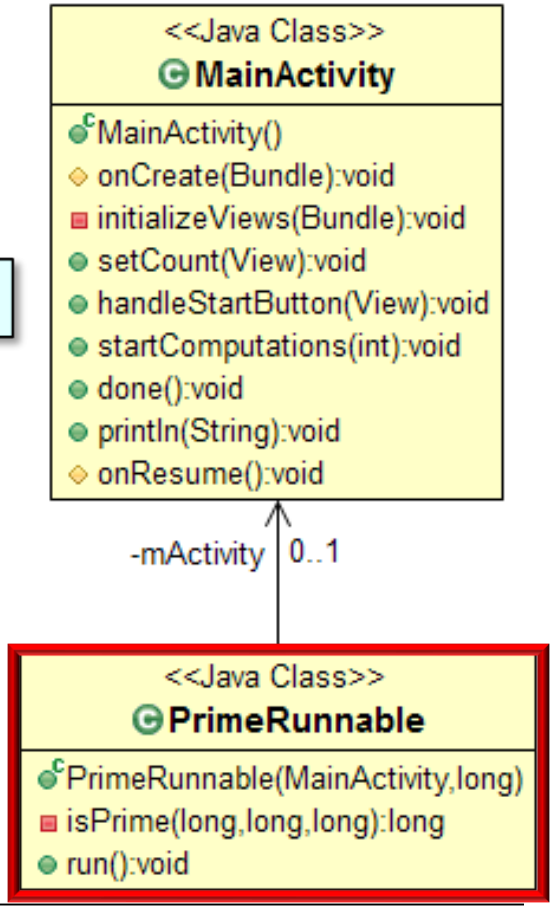
See docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html

Overview of the PrimeChecker App

- 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);
  } ...
```

Constructor stores prime # candidate & activity

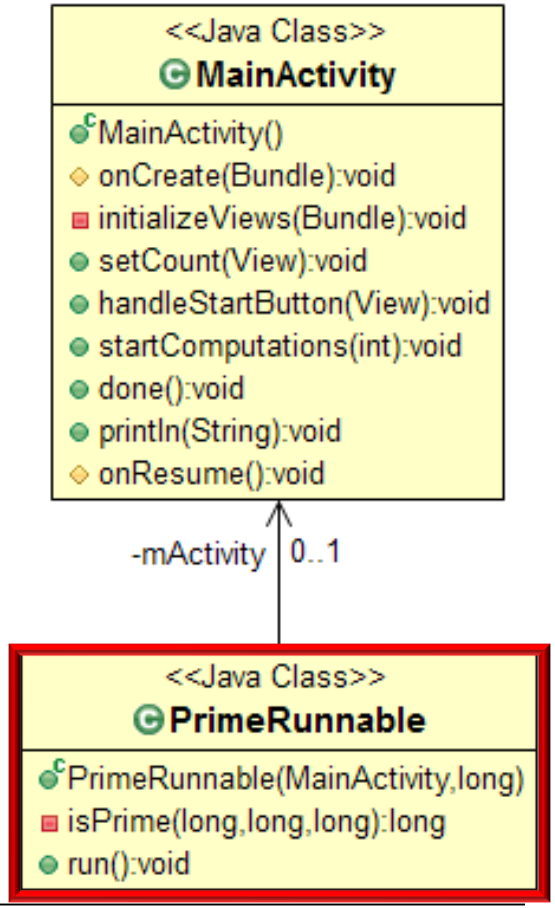


Overview of the PrimeChecker App

- 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);
  } ...
```

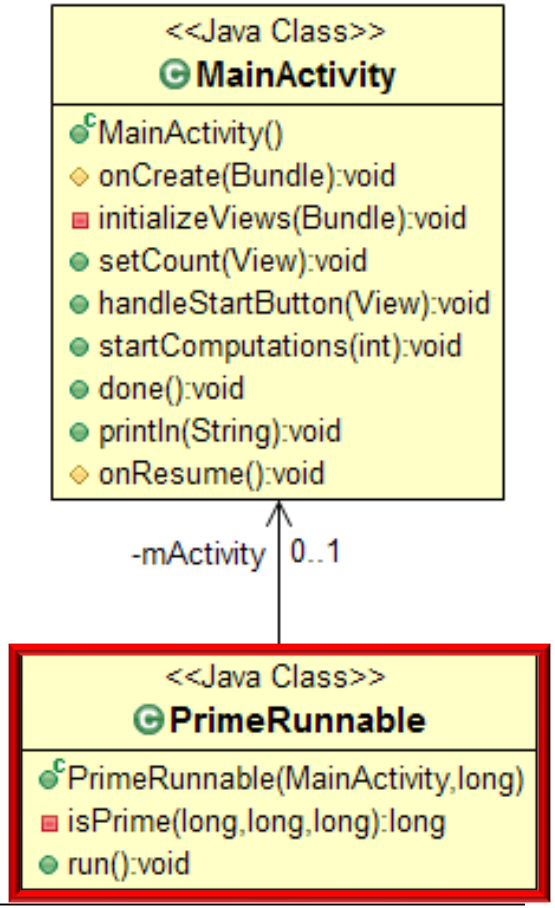
Returns 0 if n is prime or smallest factor if it's not



Overview of the PrimeChecker App

- PrimeRunnable determines if a # is prime

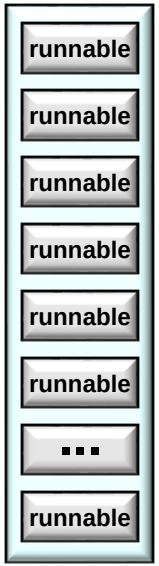
```
class PrimeRunnable implements Runnable {
  long mPrimeCandidate;
  private final MainActivity mActivity;
  ...

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

  long isPrime(long n) { ... }

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

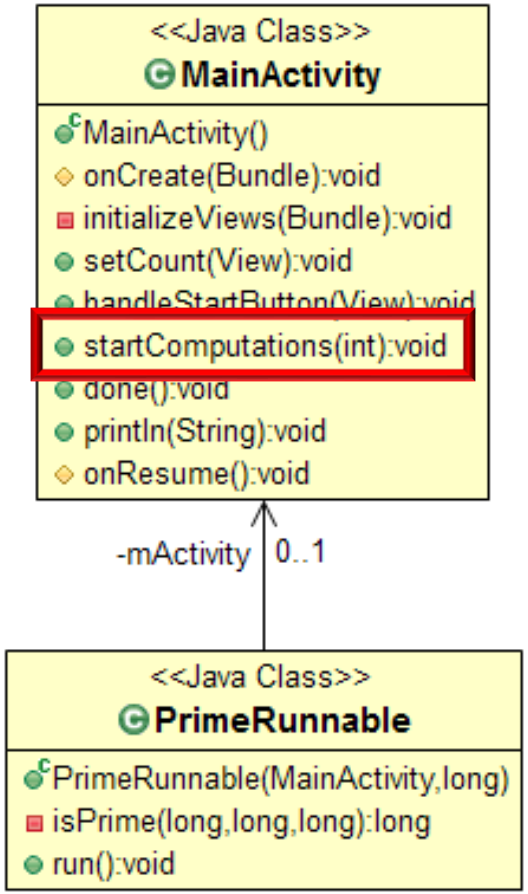


The run() hook method invokes isPrime()

Overview of the PrimeChecker App

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





Evaluating the PrimeChecker App

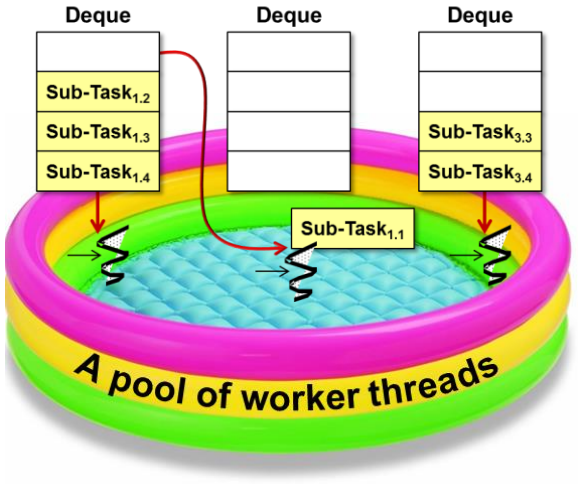
Evaluating the PrimeChecker App

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

```
new Random().longs(count, sMAX_VALUE - count, sMAX_VALUE)
            .forEach(randomNumber ->  mExecutor.execute
               (new PrimeRunnable(this, randomNumber)));
```

Fixed-sized Thread Pool

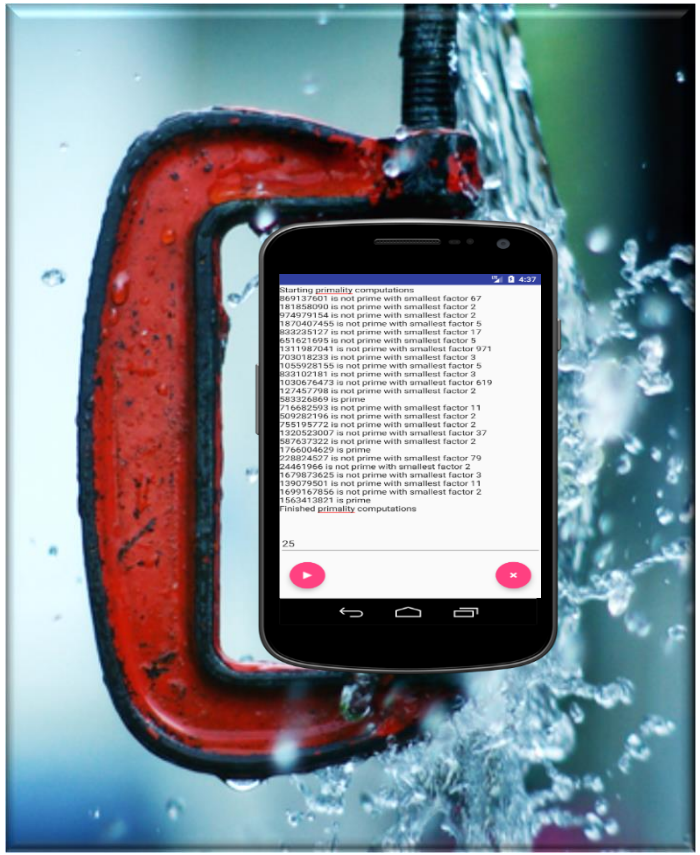
Variable-sized Thread Pool



Work-stealing Thread Pool

Evaluating the PrimeChecker App

- However, Java Executor has some restrictions



Evaluating the PrimeChecker App

- 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

Evaluating the PrimeChecker App

- 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)
        if (n / factor * factor == n)
          return factor;
    return 0;
  } ...
```

e.g., primality check is applied even if results are computed & complicates improvements

Evaluating the PrimeChecker App

- 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

Evaluating the PrimeChecker App

- 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

Evaluating the PrimeChecker App

- 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)