Java Executor:
Application to the PrimeChecker App

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

• Understand various implementation choices for the Executor interface

• Learn how to program a simple “prime checker” app using the Java Executor interface & a fixed-sized thread pool implementation
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
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

<table>
<thead>
<tr>
<th>2</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>17</td>
<td>19</td>
<td>23</td>
<td>29</td>
</tr>
<tr>
<td>31</td>
<td>37</td>
<td>41</td>
<td>43</td>
<td>47</td>
</tr>
<tr>
<td>53</td>
<td>59</td>
<td>61</td>
<td>67</td>
<td>71</td>
</tr>
<tr>
<td>73</td>
<td>79</td>
<td>83</td>
<td>89</td>
<td>97</td>
</tr>
</tbody>
</table>

See [en.wikipedia.org/wiki/Prime_number](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
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 running tasks

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

Overview of the PrimeChecker App

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

```java
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 number is prime.

```java
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](en.wikipedia.org/wiki/CPU-bound)
Overview of the PrimeChecker App

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

```java
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 fixed-sized Executor implementation

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

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newFixedThreadPool](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executors.html#newFixedThreadPool)
Overview of the PrimeChecker App

- This app uses a fixed-sized Executor implementation

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

See docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#availableProcessors
Overview of the PrimeChecker App

- This app uses a fixed-sized Executor implementation

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

*Use this value since isPrime() is a “compute-bound” task*

See [en.wikipedia.org/wiki/CPU-bound](en.wikipedia.org/wiki/CPU-bound)
Overview of the PrimeChecker App

- This app uses a fixed-sized Executor implementation

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

Stream of Random Numbers

The UI thread generates random #’s that are processed via the thread pool
Overview of the PrimeChecker App

- This app uses a fixed-sized Executor implementation

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

This fixed-size thread pool uses an unbounded queue to avoid deadlocks

See [asznaajder.github.io/thread-pool-induced-deadlocks](asznaajder.github.io/thread-pool-induced-deadlocks)

### Stream of Random Numbers

<table>
<thead>
<tr>
<th>73735</th>
<th>45963</th>
<th>78134</th>
<th>63873</th>
</tr>
</thead>
<tbody>
<tr>
<td>02965</td>
<td>58303</td>
<td>90708</td>
<td>20025</td>
</tr>
<tr>
<td>98859</td>
<td>23851</td>
<td>27965</td>
<td>62394</td>
</tr>
<tr>
<td>33666</td>
<td>62570</td>
<td>64775</td>
<td>78428</td>
</tr>
<tr>
<td>81666</td>
<td>26440</td>
<td>20422</td>
<td>05720</td>
</tr>
</tbody>
</table>
Overview of the PrimeChecker App

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

```java
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 #

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

```java
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 #

```java
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

```java
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);
        ...
    }
}
```

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) { ... }

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

See [docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html](http://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html) Overview of the PrimeChecker App

Implements Runnable
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);
        ...
    }
}
Overview of the PrimeChecker App

- PrimeRunnable determines if a # is prime

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

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

    long isPrime(long n) {
        // Returns 0 if n is prime or smallest factor if it’s not
        return smallestFactor;
    }

    void run() {
        long smallestFactor =
            isPrime(mPrimeCandidate);
        ...
    }
}
```
PrimeRunnable determines if a # is prime

```java
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
```

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.
End of Java Executor: Application to PrimeChecker App