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
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](https://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 numbers are prime.
- Each natural number 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](http://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 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

See en.wikipedia.org/wiki/CPU-bound
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;
}
```

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".
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());
```

Create fixed-sized thread pool

See 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

**UI Thread (main thread)**

Stream of Random Numbers

- Fixed WorkerThreads
- WorkQueue
- ThreadPoolExecutor

1. `execute()`
2. `offer()`
3. `take()`
4. `run()`

Returns # of processor cores known to the Java execution environment
Overview of the PrimeChecker App

- This app uses a fixed-sized Executor implementation

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

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

See 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());
```

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

Stream of Random Numbers

- UI Thread (main thread) → ThreadPoolExecutor
- Fixed WorkerThreads
- WorkQueue
- execute() → offer() → take() → run()
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 asznajder.github.io/thread-pool-induced-deadlocks
Overview of the PrimeChecker App

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

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

Diagram:
- MainActivity:
  - onCreate(Bundle):void
  - initializeViews(Bundle):void
  - setCount(View):void
  - handleStartButton(View):void
  - startComputations(int):void
  - done():void
  - println(String):void
  - onResume():void

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

`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

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 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

Implements Runnable

Overview of the PrimeChecker App
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);
        ...
    }
}
```

Constructor stores prime # candidate & activity
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
    }

    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) {
        ...
    }

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

`run()` hook method invokes `isPrime()` in a pool thread
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