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
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
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 # ‘\( N \)’
• 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 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](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](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;
}
```

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

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

```java
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 with a fixed-size thread pool tuned to the # of processor cores in the computing device.

```java
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 #

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

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](https://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 #

```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
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) {
        ...
    }

    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

```java
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 docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html
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) { ... }

    PrimeResult 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) { ... }

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

The run() hook method invokes isPrime()
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

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

---

**Fixed-sized Thread Pool**

- A pool of worker threads

**Variable-sized Thread Pool**

- A pool of worker threads

**Work-stealing Thread Pool**

- A pool of worker threads
Evaluating the PrimeChecker App

- 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(); ... 
    }
}
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
    ```java
    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)