Applying Java Executor 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

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](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 # 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)
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 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
Overview of the PrimeChecker App

- This app uses a fixed-sized Executor implementation

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

*Returns # of processor cores known to the Java execution environment*

See [docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html#availableProcessors](https://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());
```

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

Stream of Random Numbers

```
73735 45963 78134 63873
02965 58303 90708 20025
98859 23851 27965 62394
33666 62570 64775 78428
81666 26440 20422 05720
```
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 [aszajder.github.io/thread-pool-induced-deadlocks](aszajder.github.io/thread-pool-induced-deadlocks)
• 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
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 number

```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](http://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

```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 docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html
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 mMainActivity;
    ...

    PrimeRunnable(MainActivity a, Long pc) {
        mMainActivity = 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);
        ...
    }
```
PrimeRunnable determines if a number 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.
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