Java Executor: Evaluating Pros & Cons

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
- Evaluate the pros & cons of the prime checker app

Evaluating the PrimeChecker App

Evaluating the PrimeChecker App

- The Java Executor interface enables the transparent tuning & replacement of # & type of threads wrt the prime checker app logic itself

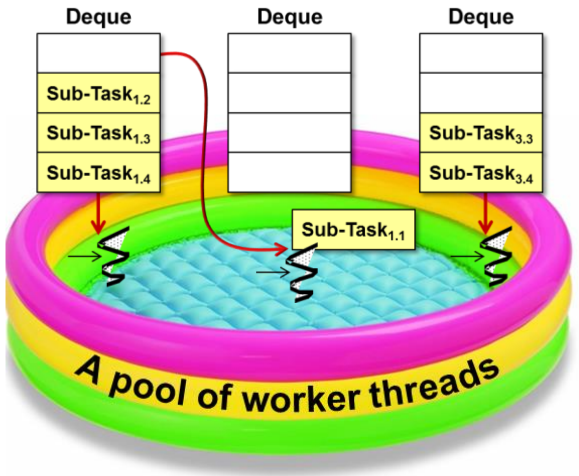
```
new Random().longs(count, sMAX_VALUE - count, sMAX_VALUE)
                .forEach(randomNumber -> mExecutor.execute
                   (new PrimeRunnable(this, randomNumber)));
```

A pool of worker threads

Fixed-sized Thread Pool

A pool of worker threads

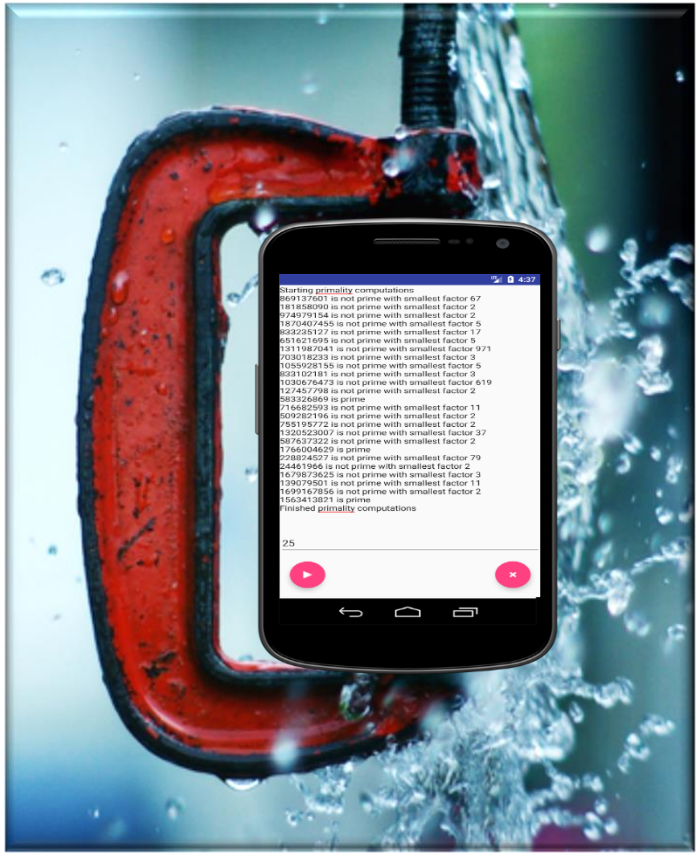
Cached (Variable-sized) Thread Pool



Work-stealing Thread Pool

Evaluating the PrimeChecker App

- However, Java Executor has some restrictions



```
Starting primality computations
869137601 is not prime with smallest factor 67
181858090 is not prime with smallest factor 2
974979154 is not prime with smallest factor 2
1870407455 is not prime with smallest factor 5
833235127 is not prime with smallest factor 17
651621695 is not prime with smallest factor 5
1311987041 is not prime with smallest factor 971
703018233 is not prime with smallest factor 3
1055928155 is not prime with smallest factor 5
833102181 is not prime with smallest factor 3
1030676473 is not prime with smallest factor 619
127457798 is not prime with smallest factor 2
583326869 is prime
716682593 is not prime with smallest factor 11
509282196 is not prime with smallest factor 2
755195772 is not prime with smallest factor 2
1320523007 is not prime with smallest factor 37
587637322 is not prime with smallest factor 2
1766004629 is prime
228824527 is not prime with smallest factor 79
24461966 is not prime with smallest factor 2
1679873625 is not prime with smallest factor 3
139079501 is not prime with smallest factor 11
1699167856 is not prime with smallest factor 2
1563413821 is prime
Finished primality computations
```

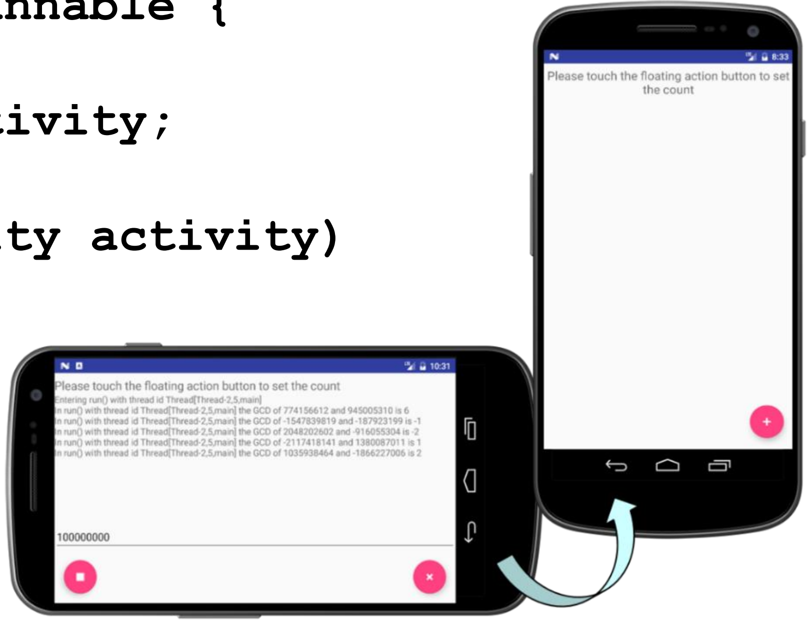
25

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(); ...
  }
}
```



Please touch the floating action button to set the count

```
Entering run() with thread id Thread[Thread-2,5,main]
In run() with thread id Thread[Thread-2,5,main] the GCD of 774156612 and 945005310 is 6
In run() with thread id Thread[Thread-2,5,main] the GCD of -1547839819 and -187923199 is -1
In run() with thread id Thread[Thread-2,5,main] the GCD of 2048202602 and -916055304 is -2
In run() with thread id Thread[Thread-2,5,main] the GCD of -2117418141 and 1380087011 is 1
In run() with thread id Thread[Thread-2,5,main] the GCD of 1035938464 and -1866227006 is 2
```

100000000

Please touch the floating action button to set the count

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., non-extensible & primality check is applied even if results are computed

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 shutdown the executor or 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 shutdown the executor or interrupt/cancel running tasks
    - Can't handle runtime configuration changes gracefully
      - e.g., must restart processing from the beginning

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 shutdown the executor or interrupt/cancel running tasks
    - Can't handle runtime configuration changes gracefully
    - The Java Executor is often too simple for its own good!

End of Java Executor: Evaluating Pros & Cons