Java ExecutorService: Evaluating Pros & Cons Douglas C. Schmidt <u>d.schmidt@vanderbilt.edu</u> 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 powerful features defined in the Java ExecutorService interface
- Understand other interfaces related to ExecutorService
- Know the key methods provided by ExecutorService
- Be aware of how ThreadPoolExecutor implements ExecutorService
- Learn how to program the PrimeChecker app using ExecutorService



• Evaluate the pros & cons of this version of the PrimeChecker app

• ExecutorService version of PrimeChecker app fixes problems with earlier Executor PrimeChecker



- ExecutorService version of PrimeChecker app fixes problems with earlier Executor PrimeChecker, e.g.
 - Two-way semantics of Java callables decouple PrimeCallable & MainActivity



public class PrimeCallable

implements Callable<PrimeResult> {

MainActivity appears nowhere in PrimeCallable class..

public PrimeCallable(long PrimeCandidate) { ... }

public PrimeResult call() {

return new PrimeResult(mPrimeCandidate,

isPrime(mPrimeCandidate));

} ...

This decoupling simplifies runtime configuration changes

- ExecutorService version of PrimeChecker app fixes problems with earlier Executor PrimeChecker, e.g.
 - Two-way semantics of Java callables decouple PrimeCallable & MainActivity
 - Lifecycle operations enable task interruptions

void interruptComputations() {
 mRetainedState.mExecutorService
 .shutdownNow();

mRetainedState.mThread.interrupt();

mRetainedState

.mExecutorService.awaitTermination

(500, TimeUnit.MILLISECONDS);



Shutting down an executor service interrupts *all* threads running tasks

223372036854775757 is not prime with smallest factor 1 223372036854775770 is not prime with smallest factor 2

23372036854775769 is not prime with smallest factor 3³ 23372036854775769 is not prime with smallest factor 3³ 23372036854775775 is not prime with smallest factor 5 23372036854775770 is not prime with smallest factor 1² 23372036854775710 is not prime with smallest factor 2

23372036854775728 is not prime with smallest factor 2 23372036854775716 is not prime with smallest factor 2 23372036854775718 is not prime with smallest factor 2 233720368547757387 is not prime with smallest factor 1 23372036854775738 is not prime with smallest factor 5

23372036854775788 is not prime with smallest factor 2 23372036854775775 is not prime with smallest factor 5 23372036854775724 is not prime with smallest factor 2

223372036854775737 is not prime with smallest factor 3 223372036854775714 is not prime with smallest factor 2 223372036854775775 is not prime with smallest factor 5 223372036854775733 is not prime with smallest factor 19

9223372036854775735 is not prime with smallest factor 5 9223372036854775748 is not prime with smallest factor 2 9223372036854775767 is not prime with smallest factor 3

0223372036854775779 is not prime with smallest factor 3 0223372036854775796 is not prime with smallest factor 2 0223372036854775771 is not prime with smallest factor 19

0223372036854775780 is not prime with smallest factor 2 0223372036854775803 is not prime with smallest factor 3 0223372036854775800 is not prime with smallest factor 2

- ExecutorService version of PrimeChecker app fixes problems with earlier Executor PrimeChecker, e.g.
 - Two-way semantics of Java callables decouple PrimeCallable & MainActivity
 - Lifecycle operations enable task interruptions

```
long isPrime(long n) {
  if (n > 3)
    for (long factor = 2;
        factor <= n / 2; ++factor)
    if (Thread.interrupted()) break;
    else if (n / factor * factor == n)
        return factor;
  return 0L;</pre>
```



ur 👂 4:37

Starting primality computations

583326869 is prime

369137601 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 551621695 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 333102181 is not prime with smallest factor 3

1030676473 is not prime with smallest factor 619 127457798 is not prime with smallest factor 2

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

- ExecutorService version of PrimeChecker app fixes problems with earlier Executor PrimeChecker, e.g.
 - Two-way semantics of Java callables decouple PrimeCallable & MainActivity
 - Lifecycle operations enable task interruptions
 - Runtime configuration changes handled gracefully



Running tasks execute & update the GUI until they finish or are interrupted

• However, there are still some limitations



- However, there are still some limitations, e.g.
 - future::get blocks the thread, even if other futures may have completed

```
private class FutureRunnable
  implements Runnable {
   MainActivity mActivity; ...
```

This problem is inherent with the "synchronous future" processing model

```
public void run() {
    mFutures.forEach(future -> {
        PrimeCallable.PrimeResult pr =
        rethrowSupplier(future::get).get();
```

```
if (pr.mSmallestFactor != 0) ...
else ...
mActivity.done(); ...
```

We fix this problem in an upcoming lesson on "Java ExecutorCompletionService"!

The "brute force" primality

- However, there are still some limitations, e.g.
 - future::get blocks the thread, even if other futures may have completed
 - isPrime() tightly coupled with PrimeCallable

```
checker always runs, even if
public class PrimeCallable ... {
                                          results were computed earlier
  long isPrime(long n) {
    if (n > 3)
      for (long factor = 2; factor <= n / 2; ++factor)
         if (Thread.interrupted())
            break;
         else if (n / factor * factor == n)
            return factor;
    return OL;
```

```
} ...
```

Fixed by Memoizer in an upcoming lesson on "Java ExecutorCompletionService"!

End of Java ExecutorService: Evaluating the Pros & Cons