Java ExecutorService: 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 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
Evaluating this Version of the PrimeChecker App
Evaluating this Version of the PrimeChecker App

- ExecutorService version of PrimeChecker app fixes problems with earlier Executor PrimeChecker
Evaluating this Version of the PrimeChecker App

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

```java
public class PrimeCallable
    implements Callable<PrimeResult> {
    ...
    public PrimeCallable(long PrimeCandidate) { ... }
    public PrimeResult call() {
        return new PrimeResult(mPrimeCandidate, isPrime(mPrimeCandidate));
    } ...
```

This decoupling simplifies runtime configuration changes
Evaluating this Version of the PrimeChecker App

- 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

```java
void interruptComputations() {
  mRetainedState.mExecutorService.shutdownNow();
  mRetainedState.mThread.interrupt();
  ...
  mRetainedState.mExecutorService.awaitTermination(500, TimeUnit.MILLISECONDS);
}
```

Shutting down an executor service interrupts all threads running tasks
Evaluator this Version of the PrimeChecker App

- 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

```java
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;
}
```

The isPrime() method repeatedly checks to see if it’s been interrupted
Evaluating this Version of the PrimeChecker App

- 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
Evaluating this Version of the PrimeChecker App

• 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

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

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

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

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

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

```java
public class PrimeCallable ... {
    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;
    }
    ...
```

The "brute force" primality checker always runs, even if results were computed earlier.

Evaluating this Version of the PrimeChecker App?

Fixed by Memoizer in an upcoming lesson on "Java ExecutorCompletionService"!
End of Java ExecutorService: Evaluating the Pros & Cons