The Java FutureTask: 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

• Understand how Java FutureTask conveys a result from a computation running in a thread to thread(s) retrieving the result
• Recognize key methods in Java FutureTask
• Know what the Memoizer class is & why it uses FutureTask to optimize programs
• Learn how to implement the Memoizer with FutureTask
• Recognize how the Memoizer class is applied to optimize prime # checking
• Evaluate the pros & cons of the Prime Checker app implementation & FutureTask
Evaluating the PrimeChecker App
Evaluating the PrimeChecker App

- The FutureTask version of the PrimeChecker app fixes a limitation with the previous version

See earlier lessons on “Java ExecutorCompletionService”
Evaluating the PrimeChecker App

- The FutureTask version of the PrimeChecker app fixes a limitation with the previous version, e.g.
- The Memoizer implementation no longer depends on ConcurrentHashMap features only available in Java 8 & beyond
Evaluating the PrimeChecker App

• However, there is still a limitation
• However, there is still a limitation, e.g.

• If the Memoizer is used for a long period of time for a wide range of inputs it will continue to grow & never clean itself up!

We fix this limitation in the lessons on the "Java ScheduledExecutorService"!
Evaluating Java FutureTask
private Future<V> computeValue(K key) {
    FutureTask<V> ft = new FutureTask<>(() -> mF.apply(key));

    Future<V> future = mCache.putIfAbsent(key, futureTask);
    if (future != null) return future;
    else { futureTask.run(); return futureTask; }
}

public V apply(final K key) {
    return mCache.computeIfAbsent(key, mFunction::apply);
}
public V apply(final K key) {
    return mCache.computeIfAbsent(key, mFunction::apply);
}
Evaluating Java FutureTask

- Java 8’s ConcurrentHashMap.computeIfAbsent() reduces need for FutureTask

```java
private Future<? extends V> computeValue(K key) {
    FutureTask<V> ft = new FutureTask<>>(() -> mF.apply(key));
    Future<V> future = mCache.putIfAbsent(key, futureTask);
    if (future != null) return future;
    else { futureTask.run(); return futureTask; }
}
```

```java
public V apply(final K key) {
    return mCache.computeIfAbsent(key, mFunction::apply);
}
```

*However, computeIfAbsent() only works if you’re using Java 8 or beyond – otherwise you’ll need to understand/use FutureTask!!*
End of Java FutureTask: Evaluating Pros & Cons