Java FutureTask: Application to Memoizer

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

Memoizer caches function call results & returns cached results for same inputs
Motivating FutureTask with a Memoizer
Motivating FutureTask with a Memoizer

- Memoization is optimization technique used to speed up programs

See [en.wikipedia.org/wiki/Memoization](en.wikipedia.org/wiki/Memoization)
Memoization is an optimization technique used to speed up programs. It caches the results of expensive function calls.

```java
V computeIfAbsent(K key, Function func) {
    1. If key doesn’t exist in cache perform a
       long-running function associated w/key
       & store the resulting value via the key
    2. Return value associated with key
}
```
Motivating FutureTask with a Memoizer

- Memoization is optimization technique used to speed up programs
- It caches the results of expensive function calls

```java
V computeIfAbsent(K key, Function func) {
    1. If key doesn’t exist in cache perform a long-running function associated w/key & store the resulting value via the key
    2. Return value associated with key
}
```
Memoization is optimization technique used to speed up programs

- It caches the results of expensive function calls

```java
V computeIfAbsent(K key, Function func) {
    1. If key doesn’t exist in cache perform a long-running function associated w/key & store the resulting value via the key
    2. Return value associated with key
}
```
Motivating FutureTask with a Memoizer

- Memoization is an optimization technique used to speed up programs.
- It caches the results of expensive function calls.
- When the same inputs occur again, the cached results are simply returned.

```java
V computeIfAbsent(K key, Function func) {
    1. If key already exists in cache
        return cached value associated w/key
}
```
Memoization is optimization technique used to speed up programs

- It caches the results of expensive function calls
- When the same inputs occur again the cached results are simply returned

```java
V computeIfAbsent(K key, Function func) {
    1. If key already exists in cache
       return cached value associated w/key
}
```
Motivating FutureTask with a Memoizer

- Memoizer defines a cache that returns a value produced by applying a (long-running) function to a key

This class is based heavily on the book “Java Concurrency in Practice” by Brian Goetz et al.
Motivating FutureTask with a Memoizer

- Memoizer defines a cache that returns a value produced by applying a (long-running) function to a key.
- A value that’s already been computed for a key is just returned, rather than applying the function to recompute it.
Memoizer defines a cache that returns a value produced by applying a (long-running) function to a key

- A value that’s already been computed for a key is just returned, rather than applying the function to recompute it

- By implementing Function a memoizer can be used whenever a Function is expected

See [docs.oracle.com/javase/8/docs/api/java/util/function/Function.html](docs.oracle.com/javase/8/docs/api/java/util/function/Function.html)
Memoizer uses a ConcurrentHashMap to minimize synchronization overhead

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html
Motivating FutureTask with a Memoizer

- Memoizer uses a ConcurrentHashMap to minimize synchronization overhead
- It uses a group of locks, each guarding a subset of the hash buckets

**ConcurrentHashMap**

Contention is low due to use of multiple locks

Motivating FutureTask with a Memoizer

- Memoizer uses a ConcurrentHashMap to minimize synchronization overhead
- It uses a group of locks, each guarding a subset of the hash buckets

**In contrast, a SynchronizedMap uses a single lock**

See codepumpkin.com/hashtable-vs-synchronizedmap-vs-concurrenthashmap
Motivating FutureTask with a Memoizer

- Memoizer uses a ConcurrentHashMap to minimize synchronization overhead
  - It uses a group of locks, each guarding a subset of the hash buckets
- computeValue() uses FutureTask to ensure a function runs only when key is first added to cache

See docs.oracle.com/javase/7/docs/api/java/util/concurrent/FutureTask.html
Motivating FutureTask with a Memoizer

- Memoizer uses a ConcurrentHashMap to minimize synchronization overhead
  - It uses a group of locks, each guarding a subset of the hash buckets
- `computeValue()` uses FutureTask to ensure a function runs only when key is first added to cache

Only one computation occurs if multiple threads simultaneously call `computeValue()` for same key
End of Java FutureTask: Application to Memoizer