SCB – Java Developer Written Assignment

Contents

[Instructions 1](#_Toc56067232)

[Questions 1](#_Toc56067233)

[Caching Function 1](#_Toc56067234)

[Deadline Scheduler 2](#_Toc56067235)

# Instructions

The role requires proven knowledge of core java development skills. The questions below are designed to allow us to gauge this ability and provide a context in the follow up interview should the you be successful. Whilst libraries may be used to aid answers, we have structured the question to allow for basic java usage. We wish to see your skills, a one liner using an external library will not give us that insight. However, if you wish to add a comment to say for production code you would use library X then we could discuss alternatives in the interview. Java 8 onwards is expected.

We are looking for production worthy code that one could promote to production without refactoring. Submissions can be in text files or via GitHub if easier and one wishes to share via that medium.

[To be replied to within 7 days]

# Questions

## Caching Function

Implement the following interface for a class that caches the results of a function.

**public interface** Cache<K, V> {  
    V get(K key);  
}

**Constraints**:

1. The implementation will return V from an internal collection if the value is cached otherwise it will call a provided Function<K, V> to get the value.
2. The implementation should allow the user of this class to provide a Function<K, V> that is used to obtain the value.
3. Important that for any unique instance of K the function is only called once.
4. How to handle null K and V is within your prerogative as is, what happens if Function<K, V> throws, however we do need to know your design choices and why in the interview.
5. Threading constraints: -
   1. The function is assumed thread-safe so for different values of K it may be called concurrently.
   2. #3 should never be violated so if 2 or more threads have a cache miss on the same key then only 1 may call the function, the other threads must wait efficiently and return the cached value once the winner has called the function and obtained a value.

The above may be implemented with a “Map.computeIfAbsent” however we are interested in how you would implement this.

## Deadline Scheduler

A component is required to schedule timer events with a given deadline.  When the deadline is met or exceeded (>=) a provided call-back interface is called with the id of the expired request.

Please read the doc of the interface methods for more in depth requirements.

The interface to implement is

*/\*\*  
\* Manages an active set of deadlines to be raised whenever they expire.  
\*/***public interface** DeadlineEngine {  
    */\*\*  
     \* Request a new deadline be added to the engine.  The deadline is in millis offset from  
     \* unix epoch. <https://en.wikipedia.org/wiki/Unix_time>  
     \* The engine will raise an event whenever a deadline (usually now in millis) supplied in the poll method   
     \* exceeds the request deadline.  
     \** ***@param deadlineMs*** *the millis  
     \** ***@return*** *An identifier for the scheduled deadline.  
     \*/* **long** schedule(**long** deadlineMs);  
  
    */\*\*  
     \* Remove the scheduled event using the identifier returned when the deadline was scheduled.  
     \** ***@param requestId*** *identifier to cancel.  
     \** ***@return*** *true if canceled.  
     \*/* **boolean** cancel(**long** requestId);  
  
    */\*\*  
     \* Supplies a deadline in millis to check against scheduled deadlines.  If any deadlines are triggered the  
     \* supplied handler is called with the identifier of the expired deadline.  
     \* To avoid a system flood and manage how many expired events we can handle we also pass in the maximum number of  
     \* expired deadlines to fire.  Those expired deadlines that wernt raised will be available in the next poll.  
     \* There is no need for the triggered deadlines to fire in order.  
     \** ***@param nowMs*** *time in millis since epoch to check deadlines against.  
     \** ***@param handler*** *to call with identifier of expired deadlines.  
     \** ***@param maxPoll*** *count of maximum number of expired deadlines to process.  
     \** ***@return*** *number of expired deadlines that fired successfully.  
     \*/* **int** poll(**long** nowMs, Consumer<Long> handler, **int** maxPoll);  
  
    */\*\*  
     \*  
     \** ***@return*** *the number of registered deadlines.  
     \*/* **int** size();  
}