**Using Callable to Return Results From Runnables**

by John Zukowski

The Runnable interface has been around since the beginning of time for the Java platform. It allows you to define a task to be completed by a thread. As most people probably know already, it offers a single method run() that accepts no arguments and returns no values, nor can it throw any checked exceptions. If you need to get a value back from the now-completed task, you must use a method outside the interface and wait for some kind of notification message that the task completed. For example, the following demonstrates what you might do for just such a scenario:

Runnable runnable = ...;

Thread t = new Thread(runnable);

t.start();

t.join();

String value = someMethodtoGetSavedValue()

Nothing is inherently wrong with this code, but it can be done differently now, thanks to the Callable interface introduced in J2SE 5.0. Instead of having a run() method, the Callable interface offers a call() method, which can return an Object or, more specifically, any type that is introduced in the genericized form:

public interface Callable<V> {

V call() throws Exception;

}

Because you cannot pass a Callable into a Thread to execute, you instead use the ExecutorService to execute the Callable object. The service accepts Callable objects to run by way of the submit() method:

<T> Future<T> submit(Callable<T> task)

As the method definition shows, submitting a Callable object to the ExecutorService returns a Future object. The get() method of Future will then block until the task is completed. This is the equivalent of the join() call in the first example. Actually, it is the equivalent of both the join() call and the get value call as get() returns the value calculated by the Callable instance.

To demonstrate, the following example creates separate Callable instances for each word passed in on the command line and sums up their length. Each Callable will just calculate the sum of its individual word. The set of Future objects are saved to acquire the calculated value from each. If the order of the returned values needed to be preserved, a List could be used instead.

import java.util.\\*;

import java.util.concurrent.\\*;

public class CallableExample {

public static class WordLengthCallable

implements Callable {

private String word;

public WordLengthCallable(String word) {

this.word = word;

}

public Integer call() {

return Integer.valueOf(word.length());

}

}

public static void main(String args[]) throws Exception {

ExecutorService pool = Executors.newFixedThreadPool(3);

Set<Future<Integer>> set = new HashSet<Future≶Integer>>();

for (String word: args) {

Callable<Integer> callable = new WordLengthCallable(word);

Future<Integer> future = pool.submit(callable);

set.add(future);

}

int sum = 0;

for (Future<Integer> future : set) {

sum += future.get();

}

System.out.printf("The sum of lengths is %s%n", sum);

System.exit(sum);

}

}

The WordLengthCallable saves each word and uses the word's length as the value returned by the call() method. This value could take some time to generate but in this case is known immediately. The only requirement of call() is the value is returned at the end of the call. When the get() method of Future is later called, the Future will either have the value immediately if the task runs quickly, as in this case, or will wait until the value is done generating. Multiple calls to get() will not cause the task to be rerun in the thread.

Because the goal of the program is to calculate the sum of all word lengths, it doesn't matter in which order the Callable tasks finish. It is perfectly OK if the last task completes before the first three. The first get() call to Future will just wait for the first task in the Set to complete. This does not block other tasks from running to completion separately. It is just waiting for that one thread or task to complete.

This particular example uses a fixed-size thread pool for the ExecutorService, but any available service will do.

**1) You have thread T1, T2 and T3, how will you ensure that thread T2 run after T1 and thread T3 run after T2?**

This thread interview questions is mostly asked in first round or phone screening round of interview and purpose of this multi-threading question is to check whether candidate is familiar with concept of "join" method or not. Answer of this multi-threading questions is simple it can be achieved by using join method of Thread class.

**2) What is the advantage of new Lock interface over synchronized block in Java? You need to implement a high performance cache which allows multiple reader but single writer to keep the integrity how will you implement it?**

The major advantage of lock interfaces on multi-threaded and concurrent programming is they provide two separate lock for reading and writing which enables you to write high performance data structure like ConcurrentHashMap and conditional blocking. This java threads interview question is getting increasingly popular and more and more follow-up questions come based upon answer of interviewee. I would strongly suggest reading Locks before appearing for any java multi-threading interview because now days Its heavily used to build cache for electronic trading system on client and exchange connectivity space.

**3) What are differences between wait and sleep method in java?**

Another frequently asked thread interview question in Java mostly appear in phone interview. Only major difference is wait release the lock or monitor while sleep doesn't release any lock or monitor while waiting. Wait is used for inter-thread communication while sleep is used to introduce pause on execution. See my post wait vs sleep in Java for more differences

**4) Write code to implement blocking queue in Java?**

This is relatively tough java multi-threading interview question which servers many purpose, it checks whether candidate can actually write Java code using thread or not, it sees how good candidate is on understanding concurrent scenarios and you can ask lot of follow-up question based upon his code. If he uses wait() and notify() method to implement blocking queue, Once interviewee successfully writes it you can ask him to write it again using new java 5 concurrent classes etc.

**5) Write code to solve the Produce consumer problem in Java?**

Similar to above questions on thread but more classic in nature, some time interviewer ask follow up questions How do you solve producer consumer problem in Java, well it can be solved in multiple way, I have shared one way to solve producer consumer problem using BlockingQueue in Java , so be prepare for surprises. Some time they even ask to implement solution of dining philosopher problem as well.

**6) Write a program which will result in deadlock? How will you fix deadlock in Java?**

Java thread interview questions and answers This is my favorite java thread interview question because even though deadlock is quite common while writing multi-threaded concurrent program many candidates not able to write deadlock free code and they simply struggle. Just ask them you have n resources and n thread and to complete an operation you require all resources. Here n can be replace with 2 for simplest case and higher number to make question more intimidating. see How to avoid deadlock in java for more information on deadlock in Java.

**7) What is atomic operation? What are atomic operations in Java?**

Simple java thread interview questions, another follow-up is do you need to synchronized an atomic operation? :) You can read more about java synchronization here.

**8) What is volatile keyword in Java? How to use it? How is it different from synchronized method in Java?**

Thread questions based on volatile keyword in Java has become more popular after changes made on it on Java 5 and Java memory model. It’s good to prepare well about how volatile variables ensures visibility, ordering and consistency in concurrent environment.

**9) What is race condition? How will you find and solve race condition?**

Another multi-threading question in Java which appear mostly on senior level interviews. Most interviewer grill on recent race condition you have faced and how did you solve it and some time they will write sample code and ask you detect race condition. See my post on Race condition in Java for more information. In my opinion this is one of the best java thread interview question and can really test the candidate's experience on solving race condition or writing code which is free of data race or any other race condition. Best book to get mastery of this topic is "Concurrency practices in Java'".

**10) How will you take thread dump in Java? How will you analyze Thread dump?**

In UNIX you can use kill -3 and then thread dump will print on log on windows you can use "CTRL+Break". Rather simple and focus thread interview question but can get tricky if he ask how you analyze it. Thread dump can be useful to analyze deadlock situations as well.

**11) Why we call start() method which in turns calls run() method, why not we directly call run() method ?**

Another classic java multi-threading interview question This was my original doubt when I started programming in thread. Now days mostly asked in phone interview or first round of interview at mid and junior level java interviews. Answer to this question is that, when you call start() method it creates new Thread and execute code declared in run() while directly calling run() method doesn’t create any new thread and execute code on same calling thread. Read my post Difference between start and run method in Thread for more details.

**12) How will you awake a blocked thread in java?**

This is tricky question on threading, blocking can result on many ways, if thread is blocked on IO then I don't think there is a way to interrupt the thread, let me know if there is any, on the other hand if thread is blocked due to result of calling wait(), sleep() or join() method you can interrupt the thread and it will awake by throwing InterruptedException. See my post How to deal with blocking methods in Java for more information on handling blocked thread.

**13) What is difference between CyclicBarriar and CountdownLatch in Java ?**

New java thread interview questions mostly to check familiarity with JDK 5 concurrent packages. One difference is that you can reuse CyclicBarrier once barrier is broken but you can not reuse ContdownLatch.

**14) What is immutable object? How does it help on writing concurrent application?**

Another classic interview questions on multi-threading, not directly related to thread but indirectly helps a lot. This java interview question can become more tricky if ask you to write an immutable class or ask you Why String is immutable in Java as follow-up.

**15) What are some common problems you have faced in multi-threading environment? How did you resolve it?**

Memory-interference, race conditions, deadlock, live lock and starvation are example of some problems comes in multi-threading and concurrent programming. There is no end of problem if you get it wrong and they will be hard to detect and debug. This is mostly experienced based interview question on java thread instead of fact based.

These were my favorite Java thread interview questions and mostly asked on Investment banks. This list is by no means complete so please contribute some of interesting java thread questions you have faced during interview. Purpose of this article is to collect and share great interview questions on multi-threading concept which not only helps on interview but opens door for learning new threading concept.

Update:

One of Javarevisited reader, Hemant has contributed some more thread interview questions in Java, though he hasn’t provide answer and left that job for me, I will certainly do when time allows, just like I have recently updated 10 Singleton interview question in Java with answers. If you guys know answers of this java concurrency questions than please post as comment:

Here is his comment “Good questions on multi-threading though you may need to prepare more in order to clear any multi-threading interview, you need to be familiar with concept of immutability, thread-safety, race condition and many more. 10 or 15 question is good for quick recap but you at-least need to prepare more than 50 questions on threading and concurrency to perform better on Java interview. You can find some interesting thread question below which is no doubt highly popular –

1) Difference between green thread and native thread in Java?

2) Difference between thread and process?

3) What is context switching in multi-threading?

4) Difference between deadlock and livelock, deadlock and starvation?

5) What thread-scheduling algorithm is used in Java?

6) What is thread-scheduler in Java?

7) How do you handle un-handled exception in thread?

8) What is thread-group, why its advised not to use thread-group in Java?

9) Why Executor framework is better than creating and managing thread by application ?

10) Difference between Executor and Executors in Java?

10) How to find which thread is taking maximum cpu in windows and Linux server?

Q)Why wait().notify() etc in Object class?

wait, notify/notifyAll methods are used by threads to communicate with each other while trying to access a common object. Putting it differently, objects become a medium via which threads communicate with each other. For example: suppose there is a 'telephone' object, which at one point of time can be used by only one thread. Like every other object in Java, 'telephone' object would also have an intrinsic lock (monitor) associated with it which at one point of time can be acquired by only one thread. Suppose the 'telephone' object requires activation before it can be used and suppose only a few admin threads can activate the 'telephone' object.  
  
As soon as a thread needs access to the 'telephone' object, it checks if the lock on that 'telephone' object is available or not, if yes, it acquires that and checks if the 'telephone' is active or not. If yes, it starts using it otherwise it calls 'wait()' on the telephone object which effectively releases the monitor of the 'telephone' object (eventually to be acquired by one of the admin threads for its activation) and puts the requester thread into the wait-set of the 'telephone' object. The requester thread goes into WAITING state. The way every object in Java has an intrinsic lock associated with it, it has an intrinsic wait-set associated with it as well.  
  
Every other non-admin requester thread goes through the same process as discussed above till one of the admin threads acquire lock on the 'telephone' object and eventually activates it and subsequently calls 'notify()' or 'notifyAll()' on the 'telephone' object. 'notify()' will simply pick one of the threads from the wait-set of the 'telephone' object (which one will be picked is an implementation dependent stuff and Java Language specification doesn't enforce any restriction on which one to be picked) and the chosen thread will now get out of WAITING mode and start trying to acquire the monitor/lock of the 'telephone' object along with any other thread that might be vying to access the 'telephone' at that point of time.

Whenever a thread acquires a lock on a any objects monitor, it becomes owner to call wait, notify etc on that object, not on the thread object. If those methods are not in Object class, then a thread can’t call these methods on an object, where the thread holds the lock.