Java ExecutorCompletionService: Implementation Internals

Douglas C. Schmidt

<u>d.schmidt@vanderbilt.edu</u>

www.dre.vanderbilt.edu/~schmidt



Professor of Computer Science

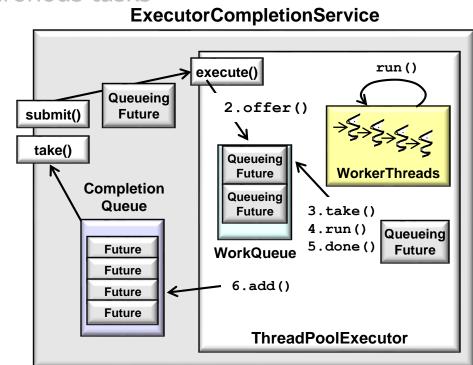
Institute for Software Integrated Systems

Vanderbilt University Nashville, Tennessee, USA



Learning Objectives in this Part of the Lesson

- Understand how the Java CompletionService interface defines a framework for handling the completion of asynchronous tasks
- Know how to instantiate the Java ExecutorCompletionService
- Recognize the key methods in the Java CompletionService interface
- Visualize the ExecutorCompletion Service in action
- Be aware of how the Java Executor CompletionService implements the CompletionService interface



 Uses an Executor to run tasks asynchronously <<Java Interface>> CompletionService<V> submit(Callable<V>) take() poll() poll(long,TimeUnit) <<Java Class>> <<Java Interface>> • ExecutorCompletionService<V> -executor Executor executor: Executor execute(Runnable):void ExecutorCompletionService(Executor) -completion new laskFor(Callable<V>) Queue submit(Callable<V>) take() <<Java Interface>> poll() BlockingQueue<E> poll(long,TimeUnit) <<Java Class>> offer(E):boolean put(E):void QueueingFuture 0..n offer(E,long,TimeUnit):boolean • take() ▲ QueueingFuture(RunnableFuture<V>) poll(long,TimeUnit) odone():void

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorCompletionService.html

- Uses an Executor to run tasks asynchronously <<Java Interface>> CompletionService<V> Results are added to a blocking queue when complete submit(Callable<V>) take() poll() poll(long,TimeUnit) <<Java Class>> <<Java Interface>> • ExecutorCompletionService<V> -executor Executor executor: Executor execute(Runnable):void
 - -completion newTaskFor(Callable<V>) Queue submit(Callable<V>) take() <<Java Interface>> poll() BlockingQueue<E> poll(long,TimeUnit) <<Java Class>> offer(E):boolean put(E):void QueueingFuture 0..n offer(E,long,TimeUnit):boolean take() ▲ QueueingFuture(RunnableFuture<V>) poll(long,TimeUnit) one():void

See src/share/classes/java/util/concurrent/ExecutorCompletionService.java

 Uses an Executor to run tasks asynchronously <<Java Interface>> CompletionService<V> Results are added to a blocking queue when complete submit(Callable<V>) take() Client threads can process these asynchronously poll() poll(long,TimeUnit) <<Java Class>> <<Java Interface>> • ExecutorCompletionService<V> -executor Executor Fexecutor: Executor execute(Runnable):void -completion newTaskFor(Callable<V>) Queue submit(Callables\/>) take() <<Java Interface>> poll() BlockingQueue<E> poll(long,TimeUnit) <<Java Class>> offer(E):boolean put(E):void QueueingFuture 0..n offer(E,long,TimeUnit):boolean • take() ▲ QueueingFuture(RunnableFuture<V>) poll(long,TimeUnit) odone():void

- There are five key methods class ExecutorCompletionService<V> implements CompletionService<V> {
 - Submit a task for execution

```
public Future<V> submit
```

RunnableFuture<V> f =

newtaskFor(task); executor.execute(new

QueueingFuture(f)); return f;

(Callable<V> task) {

(Runnable task, V result)

public Future<V> submit

{ /* ... */ } ...

- There are five key methods
 - Submit a task for execution

Remember, the futures that are

returned from these submit()

methods are typically ignored!

```
class ExecutorCompletionService<V>
implements CompletionService<V> {
```

public Future<V> submit

```
(Callable<V> task) {
   RunnableFuture<V> f =
   newtaskFor(task);
```

executor.execute(new
 QueueingFuture(f));
return f;

public Future<V> submit

{ /* ... */ } ...

(Runnable task, V result)

- There are five key methods
 - Submit a task for execution
 - Submit a two-way task



```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
 public Future<V> submit
                (Callable<V> task) {
     RunnableFuture<V> f =
       newtaskFor(task);
     executor.execute(new
             QueueingFuture(f));
     return f;
```

- There are five key methods
 - Submit a task for execution
 - Submit a two-way task

```
Provides an "async future"

processing model, where clients

don't block waiting on the future
```

QueueingFuture(f));

newtaskFor(task);

executor.execute(new

return f;

10

- There are five key methods
 - Submit a task for execution
 - Submit a two-way task

```
QueueingFuture(f));
return f;
}

public interface Callable<V> {
    V call() throws Exception;
}
```

class ExecutorCompletionService<V>

public Future<V> submit

RunnableFuture<V> f =

newtaskFor(task);

executor.execute(new

implements CompletionService<V> {

(Callable<V> task) {

- There are five key methods class ExecutorCompletionService<V>
 - Submit a task for execution.

else

Submit a two-way task

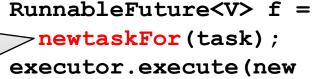
```
RunnableFuture<V> newtaskFor
```

```
(Callable<V> task) {
if (aes == null)
```



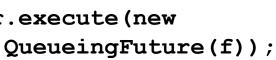
```
return new FutureTask<V>(task);
return aes.newtaskFor(task);
```



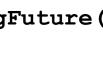


return f;

public Future<V> submit



implements CompletionService<V> {



(Callable<V> task) {



- There are five key methods class ExecutorCompletionService<V>
 - Submit a task for execution.
 - Submit a two-way task

public Future<V> submit (Callable<V> task) {

```
RunnableFuture<V> newtaskFor
         (Callable<V> task) {
```

RunnableFuture<V> f =

```
if (aes == null)
  return new FutureTask<V>(task);
else
  return aes.newtaskFor(task);
```

>newtaskFor(task); executor.execute(new QueueingFuture(f));

implements CompletionService<V> {

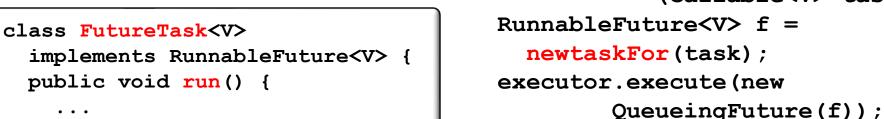
return f;

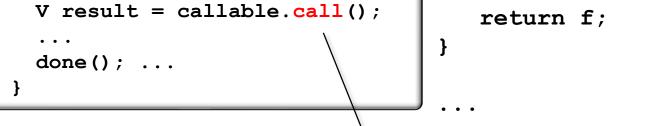
```
By default, ase.newtaskFor() encapsulates the callable task in a FutureTask
```

This default behavior can be modified by overriding the newtaskFor() method!

- There are five key methods class ExecutorCompletionService<V>
 - Submit a task for execution
 - Submit a two-way task

implements CompletionService<V> {





FutureTask's run() hook method invokes the task's call() method

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html

- There are five key methods class ExecutorCompletionService<V>
 - Submit a task for execution
 - Submit a two-way task

```
implements CompletionService<V> {
    ...
public Future<V> submit
```

alls the done() hook method if all goes well

There are five key methods

void run();

- Submit a task for execution
 - Submit a two-way task

RunnableFuture's run() hook method must be overridden by a subclass

RunnableFuture<V> f =

newtaskFor(task);

executor.execute(new

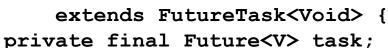
return f;

class ExecutorCompletionService<V>

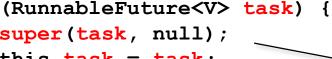
QueueingFuture(f));

- There are five key methods
 - Submit a task for execution.
 - Submit a two-way task

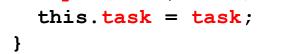
```
class QueueingFuture
```



QueueingFuture



```
this.task = task;
```

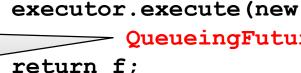


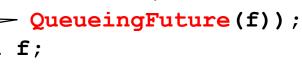
protected void done()

super(task, null);

completionQueue.add(task); }







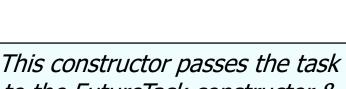
class ExecutorCompletionService<V>

public Future<V> submit

RunnableFuture<V> f =

newtaskFor(task);

implements CompletionService<V> {



(Callable<V> task) {

to the FutureTask constructor & stores the task in a future field

See src/share/classes/java/util/concurrent/ExecutorCompletionService.java

- There are five key methods class ExecutorCompletionService<V>
 - Submit a task for execution
 - Submit a two-way task

```
public Future<V> submit
```

newtaskFor(task);

executor.execute(new

implements CompletionService<V> {

(Callable<V> task) {

RunnableFuture<V> f =

```
class QueueingFuture
    extends FutureTask<Void> {
  private final Future<V> task;
  QueueingFuture
    (RunnableFuture<V> task) {
    super(task, null);
    this.task = task;
```

```
QueueingFuture(f));
return f;
```

protected void done()
{ completionQueue.add(task); }

**This done() hook method adds the
future to the queue upon completion

See src/share/classes/java/util/concurrent/ExecutorCompletionService.java

- There are five key methods
 - Submit a task for execution
 - Submit a two-way task
 - Submit a one-way task



```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
 public Future<V> submit
                (Callable<V> task) {
 public Future<V> submit
           (Runnable task, V result)
  { /* ... */ }
```

- There are five key methods
 - Submit a task for execution
 - Retrieve results

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
  public Future<V> take() ...
   { return completionQueue.take(); }
  public Future<V> poll()
   { return completionQueue.poll(); }
  public Future<V> poll(long
     timeout, TimeUnit unit) ... {
     return completionQueue.poll
```

(timeout, unit);

- There are five key methods
 - Submit a task for execution
 - Retrieve results
 - Block until a future for next completed task is available
 - Then retrieve/remove it

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
    ...
    public Future<V> take() ...
    { return completionQueue.take(); }
    public Future<V> poll()
```

public Future<V> poll(long

(timeout, unit);

{ return completionQueue.poll(); }

timeout, TimeUnit unit) ... {

return completionQueue.poll

- There are five key methods
 - Submit a task for execution
 - Retrieve results
 - Block until a future for next completed task is available
 - Retrieve/remove a future for the next completed task
 - Returns null if no future is available

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
   public Future<V> take() ...
     return completionQueue.take(); }
   public Future<V> poll()
   { return completionQueue.poll(); }
   public Future<V> poll(long
     timeout, TimeUnit unit) ... {
     return completionQueue.poll
       (timeout, unit);
```

- There are five key methods
 - Submit a task for execution
 - Retrieve results
 - Block until a future for next completed task is available
 - Retrieve/remove a future for the next completed task
 - Wait up to specified time if future isn't available
 - Returns null if timeout occurs

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
  public Future<V> take()
    return completionQueue.take(); }
  public Future<V> poll()
   { return completionQueue.poll(); }
  public Future<V> poll(long
     timeout, TimeUnit unit) ... {
     return completionQueue.poll
       (timeout, unit);
```

 Allows 1+ client threads to submit **ExecutorCompletionService** two-way tasks to a pool, while 1+ run() other threads process async results execute() Queueing 2.offer() submit() **Future** 1.submit(task) take() Queueina WorkerThreads **Future** Completion Queueina Queue 3.take() **Future** 4.run() Queueing 5.done() **Future Future** 7. take() WorkQueue **Future** 6.add() **Future** Future **ThreadPoolExecutor**

End of Java Executor CompletionService: Implementation Internals