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 ExecutorCompletionService in action
- Be aware of how the Java Executor CompletionService implements the CompletionService interface
Implementation of the Java ExecutorCompletionService
Implementation of the ExecutorCompletionService

- Uses an Executor to run tasks asynchronously

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

- Uses an Executor to run tasks asynchronously
- Results are added to a blocking queue when complete

See openjdk/6-b14/java/util/concurrent/ExecutorCompletionService.java
Implementation of the ExecutorCompletionService

- Uses an Executor to run tasks asynchronously
- Results are added to a blocking queue when complete
- Client threads can process these asynchronously
Implementation of the ExecutorCompletionService

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

```java
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) {
        /* ... */
    }

    /* ... */
```
Implementation of the ExecutorCompletionService

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

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

    public Future<V> submit (Runnable task, V result) {
        /* ... */
    }
}
```

Remember, the futures that are returned from these submit() methods are typically ignored!
Implementation of the ExecutorCompletionService

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

```java
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;
    }
    ...
```
Implementation of the ExecutorCompletionService

- 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

```
class ExecutorCompletionService<
  implements CompletionService<
  {
    
    public Future< submit
      (Callable<
        RunnableFuture< 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

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

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html
Implementation of the ExecutorCompletionService

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

```java
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;
  }
  ...
}
```

RunnableFuture<V> newtaskFor (Callable<V> task) {
  ...
  return new FutureTask<V> (task);
  ...
}
Implementation of the ExecutorCompletionService

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

```
RunnableFuture<V> newtaskFor
    (Callable<V> task) {
    ...
    return new FutureTask<V>(task);
    ...
}
```

The callable task is encapsulated in a FutureTask
Implementation of the ExecutorCompletionService

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

```java
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;
    }
    ...
}
```

```java
class FutureTask<V> implements RunnableFuture<V> {
    public void run() {
        ... 
        V result = callable.call();
        ...
        done(); ... 
    }
}
```

*FutureTask’s run() hook method invokes the task’s call() method*

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html)
Implementation of the ExecutorCompletionService

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

```java
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;
    }
    ...
}
```

FutureTask's run() hook method also calls the done() hook method if all goes well

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/FutureTask.html)
Implementation of the ExecutorCompletionService

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

```java
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

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/RunnableFuture.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/RunnableFuture.html) for more information.

RunnableFuture’s run() hook method must be overridden by a subclass.
Implementation of the ExecutorCompletionService

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

```java
class QueueingFuture
    extends FutureTask<Void> {
    private final Future<V> task;
    QueueingFuture
        (RunnableFuture<V> task) {
        super(task, null);
        this.task = task;
    }
    protected void done() {
        completionQueue.add(task);
    }
}
```

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

See `src/share/classes/java/util/concurrent/ExecutorCompletionService.java`
Implementation of the ExecutorCompletionService

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

```java
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;
    }
    ...
}
```

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

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

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

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

```java
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

Implementation of the ExecutorCompletionService

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
- Returns null if no future is available

```java
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);
  }
  ...
}
Implementation of the ExecutorCompletionService

- Allows 1+ client threads to process two-way tasks in a pool, while 1+ other threads process results.

1. submit(task)
2. offer()
3. take()
4. run()
5. done()
6. add()
7. take()
End of Java Executor
CompletionService:
Implementation Internals