Key Methods in the Java ExecutorCompletionService

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 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

```java
<<Java Interface>>
CompletionService<V>

- submit(Callable<V>)
- submit(Runnable, V)
- take()
- poll()
- poll(long, TimeUnit)
```
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
- These methods are implemented by the ExecutorCompletionService class
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
Key Methods in the CompletionService Interface
Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods

**Interface CompletionService<V>**

All Known Implementing Classes:
ExecutorCompletionService

```java
public interface CompletionService<V>
```

A service that decouples the production of new asynchronous tasks from the consumption of the results of completed tasks. Producers submit tasks for execution. Consumers take completed tasks and process their results in the order they complete. A CompletionService can for example be used to manage asynchronous I/O, in which tasks that perform reads are submitted in one part of a program or system, and then acted upon in a different part of the program when the reads complete, possibly in a different order than they were requested.

Typically, a CompletionService relies on a separate Executor to actually execute the tasks, in which case the CompletionService only manages an internal completion queue. The ExecutorCompletionService class provides an implementation of this approach.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionService.html](docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionService.html)
The CompletionService interface only defines a few methods, e.g.

- Submit a task for execution

```java
class ExecutorCompletionService&lt;V&gt; implements CompletionService&lt;V&gt; {
    ...
    public Future&lt;V&gt; submit(Callable&lt;V&gt; task) {
        ...
    }
    public Future&lt;V&gt; submit(Runnable task, V result) {
        ...
    }
    ...
}
```
The CompletionService interface only defines a few methods, e.g.

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

```java
class ExecutorCompletionService<V> implements CompletionService<V> {
    ...
    public Future<V> submit(Callable<V> task) {
        ...
    }

    public Future<V> submit(Runnable task, V result) {
        ...
    }
    ...
}
```
Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
  - Submit a value-returning two-way task

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

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

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html
The CompletionService interface only defines a few methods, e.g.

- Submit a task for execution
- Submit a value-returning two-way task
- Provides an “asynchronous future” processing model

### Key Methods in the CompletionService Interface

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

    Return values of submit() are typically ignored
```

i.e., no need to block on the future
# Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
  - Submit a value-returning two-way task
  - Provides an "asynchronous future" processing model
  - The main reason to access this future is to cancel the async computation

```java
class ExecutorCompletionService<V> implements CompletionService<V> {
    ...
    public Future<V> submit(Callable<V> task) {
        ...
    }

    public Future<V> submit(Runnable task, V result) {
        ...
    }
}
```
The CompletionService interface only defines a few methods, e.g.

- Submit a task for execution
  - Submit a value-returning two-way task
- Submit a one-way task that returns nothing

```
class ExecutorCompletionService<V>
  implements CompletionService<V> {
    ...
    public Future<V> submit(Callable<V> task) {
      ...
    }

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

Not as widely used as the two-way callable task
The `CompletionService` interface only defines a few methods, e.g.

- Submit a task for execution
  - Submit a value-returning two-way task
- Submit a one-way task that returns nothing

```java
public interface Runnable {
    void run();
}
```

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

See [docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html](http://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html)
The CompletionService interface only defines a few methods, e.g.

- Submit a task for execution
- Retrieve results

These methods access an internal blocking queue containing Queueing Futures whose tasks have completed.

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

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html
Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
  - Retrieve results

*get()* never blocks on a future removed from the internal queue!

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

Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution
  - Retrieve results
  - Block until a future for next completed task is available & then retrieve/remove it

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

    public Future<V> take() ... {
        ...
    }

    public Future<V> poll() {
        ...
    }

    public Future<V> poll(long timeout, TimeUnit unit) ... {
        ...
    }
    ...
```
The CompletionService interface only defines a few methods, e.g.

- Submit a task for execution
- Retrieve results
- Block until a future for next completed task is available & then retrieve/remove it
- Retrieve/remove a future for the next completed task or null if none are available

```java
class ExecutorCompletionService<V> implements CompletionService<V> {
    ...
    public Future<V> take() ... {
        ...
    }
    public Future<V> poll() {
        ...
    }
    public Future<V> poll(long timeout, TimeUnit unit) ... {
        ...
    }
    ...
}
```
The CompletionService interface only defines a few methods, e.g.

- Submit a task for execution
- Retrieve results
  - Block until a future for next completed task is available & then retrieve/remove it
  - Retrieve/remove a future for the next completed task or null if none are available
  - Block up to the specified wait time if future isn’t available

class ExecutorCompletionService<V> implements CompletionService<V> {
  ...
  public Future<V> take() ... {
    ...
    }
  
  public Future<V> poll() {
    ...
    }
  
  public Future<V> poll(long timeout, TimeUnit unit) ... {
    ...
    }
  ...
}
Visualizing the Java ExecutorCompletionService
ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete.

Visualizing the Java ExecutorCompletionService:

1. `submit(task)`

2. `offer()`

3. `take()`

4. `run()`

5. `done()`

6. `add()`

7. `take()`

1+ threads submit two-way tasks to a thread pool, while 1+ threads handle results of these tasks.
Visualizing the Java ExecutorCompletionService

- ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete.

A client submits a two-way task
Visualizing the Java ExecutorCompletionService

- ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete.

The task is encapsulated in a QueueingFuture & enqueued for subsequent worker thread processing.
Visualizing the Java ExecutorCompletionService

- ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete.

A worker thread in the thread pool dequeues a queueing future & runs it.
Visualizing the Java ExecutorCompletionService

- ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete.

When queueing future is finished running its result is added to the completion queue for later processing.
Visualizing the Java ExecutorCompletionService

- ExecutorCompletionService uses an Executor to run tasks, which are then added to its internal blocking queue when they complete.

A client thread gets completed tasks from completion queue & then processes them.
End of Key Methods in the Java ExecutorCompletionService