The Java CompletionService
(Part 1)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

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
Motivating the Java CompletionService Interface
One problem with the ExecutorService implementation of the PrimeChecker app is that submit() returned a future that must be handled synchronously.

Motivating the Java CompletionService Interface...

private class FutureRunnable implements Runnable {
    List<Future<PrimeCallable.PrimeResult>> mFutures;
    MainActivity mActivity; ...

    public void run() {
        for (Future<PrimeResult> f : mFutures) {
            ...
            PrimeResult pr = f.get();
            if (pr.mSmallestFactor != 0) ...
            else ...
            ...
            mActivity.done();
        }
    }

f.get() blocks the thread, even if some other futures may have completed

This is a common problem with the “synchronous future processing” model.
Motivating the Java CompletionService Interface

- The CompletionService fixes this problem by supporting an “asynchronous future processing model”

As two-way tasks complete their results are stored in a completion queue & can be processed immediately in one or more client threads
Overview of the Java CompletionService Interface
Overview of the Java CompletionService Interface

- The CompletionService decouples async task invocation from the processing of completed task results.

See [docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionService.html](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionService.html)
Overview of the Java CompletionService Interface

- The CompletionService decouples async task invocation from the processing of completed task results
- Implemented via the ExecutorCompletionService
Overview of the Java CompletionService Interface

- The CompletionService decouples async task invocation from the processing of completed task results
- Implemented via the ExecutorCompletionService

```
<<Java Interface>>
CompletionService<V>
submit(Callable<V>)
take()
poll()
poll(long, TimeUnit)
```

```
<<Java Interface>>
ExecutorService
submit(Callable<T>)
invokeAll(Collection<? extends Callable<T>>)?
invokeAny(Collection<? extends Callable<T>>)?
shutdown(): void
shutdownNow()
awaitTermination(long, TimeUnit): boolean
```

```
<<Java Interface>>
BlockingQueue<E>
offer(E): boolean
put(E): void
offer(E, long, TimeUnit): boolean
take()
poll(long, TimeUnit)
```

```
<<Java Class>>
ExecutorCompletionService<V>
executor: Executor
completionQueue: BlockingQueue<Future<V>>
newTaskFor(Callable<V>)
submit(Callable<V>)
take()
poll()
poll(long, TimeUnit)
```

```
<<Java Class>>
QueueingFuture
future: Future<V>
QueueingFuture(RunnableFuture<V>)
done(): void
```
Overview of the Java CompletionService Interface

- The CompletionService decouples async task invocation from the processing of completed task results
- Implemented via the ExecutorCompletionService

An Executor is used to execute tasks in a pool of threads
The CompletionService decouples async task invocation from the processing of completed task results. It is implemented via the ExecutorCompletionService.

### Overview of the Java CompletionService Interface

- **CompletionService<V>**
  - submit(Callable<V>)
  - take()
  - poll()
  - poll(long, TimeUnit)

- **ExecutorCompletionService<V>**
  - executor: Executor
  - completionQueue: BlockingQueue<Future<V>>

- **ExecutorService**
  - submit(Callable<T>)
  - invokeAll(Collection<? extends Callable<T>>)
  - invokeAny(Collection<? extends Callable<T>>)
  - shutdown(): void
  - shutdownNow(): void
  - awaitTermination(long, TimeUnit): boolean

- **BlockingQueue<E>**
  - offer(E): boolean
  - put(E): void
  - offer(E, long, TimeUnit): boolean
  - take():
  - poll(long, TimeUnit)

- **QueueingFuture**
  - task: Future<V>
  - newTaskFor(Callable<V>)
  - submit(Callable<V>)
  - take():
  - poll():
  - poll(long, TimeUnit)

Completed tasks are put on a blocking queue that's accessed via take()/poll().
The CompletionService decouples async task invocation from the processing of completed task results.

Implemented via the ExecutorCompletionService.
Overview of the Java CompletionService Interface

- CompletionService implements the *Proactor* pattern
- Supports the demultiplexing & dispatching of multiple event handlers, which are triggered by the completion of asynchronous events

See [en.wikipedia.org/wiki/Proactor_pattern](en.wikipedia.org/wiki/Proactor_pattern)
End of Overview of the Java CompletionService (Part 1)
The Java CompletionService
(Part 2)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

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.
- Recognize the key methods in the Java CompletionService interface.

```
<<Java Interface>>
CompletableFuture<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
- Recognize the key methods in the Java CompletionService interface
- Know how the ExecutorCompletionService implements CompletionService
Key Methods in the CompletionService Interface
Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods

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

- The CompletionService interface only defines a few methods, e.g.
- Submit a task for execution & return a future

```java
public 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 & return a future
- Submit a value-returning task

```java
public 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 & return a future
- Submit a value-returning task

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

```java
public 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](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Callable.html)
Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution & return a future
  - Submit a value-returning task
  - Provides an “asynchronous future” processing model

```java
public 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
Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution & return a future
    - Submit a value-returning task
    - Submit a runnable task

```java
public 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
Key Methods in the CompletionService Interface

- The CompletionService interface only defines a few methods, e.g.
  - Submit a task for execution & return a future
    - Submit a value-returning task
    - Submit a runnable task

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

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

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

See docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html
Key Methods in the CompletionService Interface

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

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

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

    public Future<V> poll(long timeout, TimeUnit unit) ... {
        ...
    }
    ...
}
```

These three methods all access an internal completion queue that contains instances of QueueFutures whose tasks have completed.
Key Methods in the CompletionService Interface

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

```java
public 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 & return a future
  - 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
public 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 & return a future
  - 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
    - Like poll(), but wait up to the specified wait time if future isn’t available

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

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

    public Future<V> poll(long timeout, TimeUnit unit) ...
    {
        ...
    }
    ...
```
Overview of the Java ExecutorCompletionService
Overview of the ExecutorCompletionService

• The ExecutorCompletionService implements the CompletionService & uses an Executor to execute tasks placed on a blocking queue when they complete.

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

• A program typically creates an Executor (or ExecutorService) instance & then associates it with a new ExecutorCompletionService

```java
// Create a thread pool that matches the number of cores.
mExecutorService =
    Executors.newFixedThreadPool(Runtime.getRuntime()
        .availableProcessors());

// Associate the ExecutorCompletionService with the ExecutorService.
mExecutorCompletionService =
    new ExecutorCompletionService<>(mExecutorService);
```
Overview of the `ExecutorCompletionService`

- The `ExecutorCompletionService` uses its associated `Executor` to execute tasks placed on a blocking queue when they complete.

![Diagram of ExecutorCompletionService]

1. `submit(task)`
2. `offer()`
3. `take()`
4. `run()`
5. `done()`
6. `add()`
7. `take()`

Allows one or more worker threads to process two-way tasks in a thread pool, while one or more other threads process tasks results asynchronously.
Overview of the ExecutorCompletionService

- The ExecutorCompletionService uses its associated Executor to execute tasks placed on a blocking queue when they complete.

A client submits a two-way callable task via the ExecutorCompletionService.
Overview of the ExecutorCompletionService

- The ExecutorCompletionService uses its associated Executor to execute tasks placed on a blocking queue when they complete.

This task is converted into a QueueingFuture & enqueued onto a work queue for subsequent process via a thread in the worker thread pool.
Overview of the ExecutorCompletionService

• The ExecutorCompletionService uses its associated Executor to execute tasks placed on a blocking queue when they complete.

```
submit()  
Queueing Future  
execute()  
Queueing Future  
WorkerThreads  
run()  
Future  
Future  
Future  
Future
```

A worker thread in the thread pool dequeues the queueing future & runs it.
Overview of the ExecutorCompletionService

- The ExecutorCompletionService uses its associated Executor to execute tasks placed on a blocking queue when they complete.

When the queueing future is finished running the result is added to the Completion Queue for subsequent processing.
Overview of the ExecutorCompletionService

- The ExecutorCompletionService uses its associated Executor to execute tasks placed on a blocking queue when they complete.

```
submit()
```

```
take()
```

```
run()
```

A client thread can then retrieve each completed task from the CompletionQueue & process it.
End of Overview of the Java CompletionService (Part 2)