Java ExecutorCompletionService: Introduction

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 Java CompletionService’s interface defines a framework for handling the completion of async tasks

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

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

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

<<Java Class>>
1 QueueingFuture
  - task: Future<V>
  - QueueingFuture(RunnableFuture<V>)
  - done():void
```
Learning Objectives in this Part of the Lesson

• Understand how Java CompletionService’s interface defines a framework for handling the completion of async tasks

• Know how to instantiate the Java ExecutorCompletionService

```java
mExecutorService = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());

mExecutorCompletionService = new ExecutorCompletionService<>(mExecutorService);
```
Motivating the Java CompletionService Interface
• One problem with the ExecutorService implementation of the PrimeChecker app is that the future submit() returned must be handled synchronously

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

    public void run() {
        mFutures.forEach(future -> {
            PrimeCallable.PrimeResult pr = rethrowSupplier(future::get).get();
        ...
```

This blocking problem is common w/the “synchronous future” processing model.
Motivating the Java CompletionService Interface

- CompletionService fixes this problem via an “asynchronous future” processing model

Two-way task results are stored in a completion queue & can be processed immediately
Overview of the Java CompletionService Interface
Overview of the Java CompletionService Interface

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

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionService.html
The CompletionService interface decouples async task invocation from the processing of completed task results. Implemented via the ExecutorCompletionService class.

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

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

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

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

**Java Class**
```
Executor
```
- execute(Runnable):void

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

**Java Class**
```
QueueingFuture
```
- task: Future<V>
- QueueingFuture(RunnableFuture<V>)
- done():void

*which contains an Executor & a BlockingQueue*
Overview of the Java CompletionService Interface

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

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

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

Completed tasks are put blocking queue accessed via take()/poll()
Overview of the Java CompletionService Interface

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

Extends FutureTask to queue a task when it’s “done”
• CompletionService can implement the *Proactor* pattern

Supports demultiplexing & dispatching of event handlers that are triggered by the completion of async events

```
handle_event (Event event)
begin
  ## Process the received event
  if (event.type == REQUEST)
    ## Read request asynchronously
    ## and return control.
    async_read();
  elseif (event.type == READ_COMPLETE)
    ## Process event, deliver results
    ## asynchronously, and return control.
    process_data();
    async_write();
  end

end
```

See [en.wikipedia.org/wiki/Proactor_pattern](http://en.wikipedia.org/wiki/Proactor_pattern)
Instantiating the Java ExecutorCompletionService
Instantiating the Java ExecutorCompletionService

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

```java
mExecutorService = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
```

```java
mExecutorCompletionService = new ExecutorCompletionService<>(mExecutorService);
```
• A program typically creates an Executor (or ExecutorService) instance & then associates it with a new ExecutorCompletionService

```java
mExecutorService = Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors());
```

Create an executor service whose thread pool size matches the # of cores

```java
mExecutorCompletionService = new ExecutorCompletionService<>(mExecutorService);
```
A program typically creates an Executor (or ExecutorService) instance & then associates it with a new ExecutorCompletionService

\[
m_{ExecutorService} = \text{Executors.newFixedThreadPool} \left( \text{Runtime.getRuntime().availableProcessors()} \right);
\]

\[
m_{ExecutorCompletionService} = \text{new ExecutorCompletionService<<}(m_{ExecutorService});
\]
End of the Java Executor CompletionService: Introduction