Java ExecutorCompletionService: Introduction

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 Java CompletionService's interface defines <<Java Interface>> CompletionService<V> a framework for handling the completion of async tasks submit(Callable<V>) take() poll() poll(long,TimeUnit) <<Java Class>> <<Java Interface>> • ExecutorCompletionService<V> -executor Executor Fexecutor: 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) odone():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

Motivating the Java CompletionService Interface

Motivating the Java CompletionService Interface

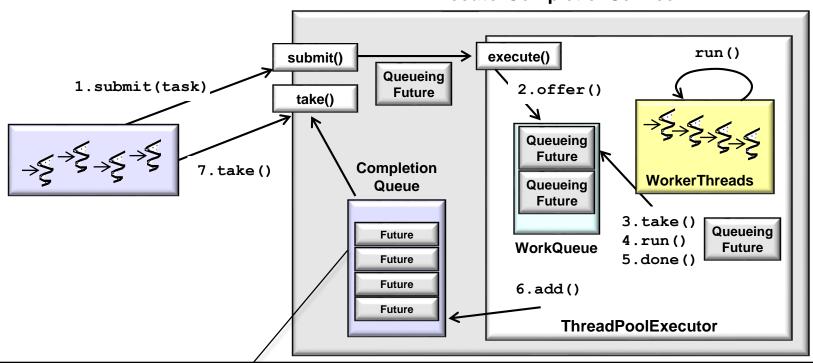
 One problem with the ExecutorService implementation of the PrimeChecker app is that the future submit() returned must be handled synchronously

```
private class FutureRunnable
  implements Runnable {
  List<Future<PrimeCallable.PrimeResult>>
    mFutures;
  MainActivity mActivity; ...
                                             future::get may block the
                                             thread, even if some other
 public void run() {
                                            futures may have completed
   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
 ExecutorCompletionService



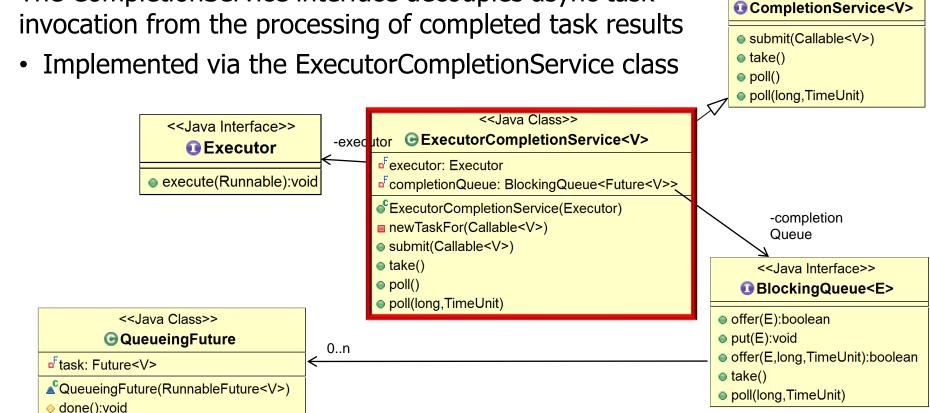
Two-way task results are stored in a completion queue & can be processed immediately

 The CompletionService interface decouples async task <<Java Interface>> CompletionService<V> invocation from the processing of completed task results 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) odone():void

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

<<Java Interface>>

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



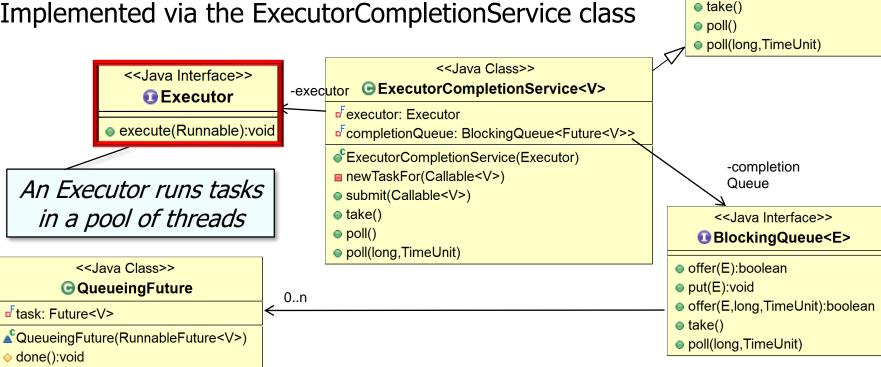
See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorCompletionService.html

 The CompletionService interface decouples async task <<Java Interface>> CompletionService<V> invocation from the processing of completed task results submit(Callable<V>) Implemented via the ExecutorCompletionService class take() poll() poll(long,TimeUnit) <<Java Class>> <<Java Interface>> • ExecutorCompletionService<V> -executor Executor Fexecutor: 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 which contains an Executor • take() ▲ QueueingFuture(RunnableFuture<V>) poll(long,TimeUnit) & a BlockingQueue odone():void

<<Java Interface>> CompletionService<V>

submit(Callable<V>)

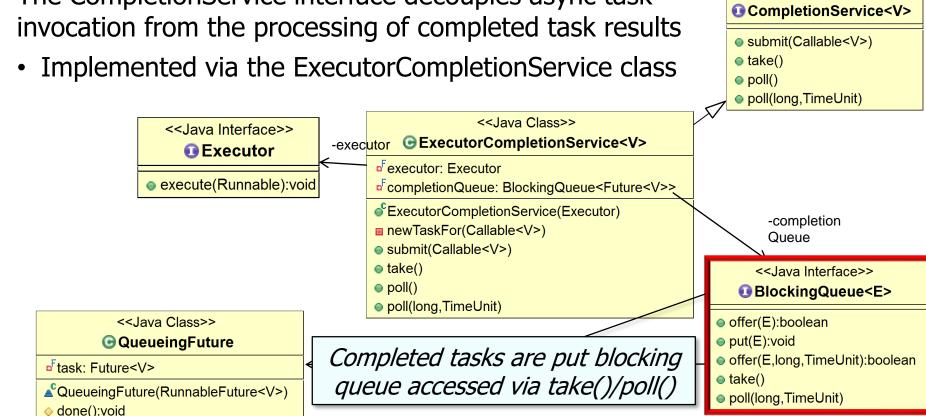
- The CompletionService interface decouples async task invocation from the processing of completed task results
 - Implemented via the ExecutorCompletionService class



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

<<Java Interface>>

The CompletionService interface decouples async task



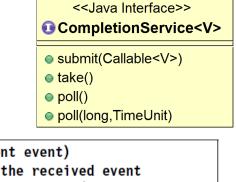
See docs.oracle.com/javase/8/docs/api/java/util/concurrent/BlockingQueue.html

- The CompletionService interface decouples async task <<Java Interface>> CompletionService<V> invocation from the processing of completed task results submit(Callable<V>) Implemented via the ExecutorCompletionService class take() poll() poll(long,TimeUnit) <<Java Class>> <<Java Interface>> • ExecutorCompletionService<V> -executor Executor Fexecutor: 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 Extends FutureTask to queue offer(E,long,TimeUnit):boolean • take() a task when it's "done" ▲ QueueingFuture(RunnableFuture<V>) poll(long,TimeUnit) one():void

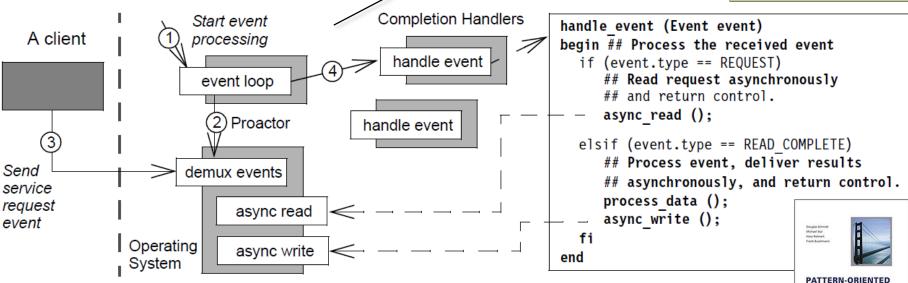
See src/share/classes/java/util/concurrent/ExecutorCompletionService.java

CompletionService can implement the *Proactor* pattern

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



SOFTWARE
ARCHITECTURE
Patterns for Concurrent



See en.wikipedia.org/wiki/Proactor_pattern

 ExecutorCompletionService implements CompletionService <<Java Interface>> CompletionService<V> & uses an Executor to execute tasks placed on a blocking submit(Callable<V>) queue when they complete take() poll() poll(long,TimeUnit) <<Java Class>> <<Java Interface>> • ExecutorCompletionService<V> -executor Executor Fexecutor: 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) odone():void

See docs.orade.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

```
associates it with a new ExecutorCompletionService
mExecutorService =
    Executors.newFixedThreadPool(Runtime.getRuntime())
```

.availableProcessors());

```
mExecutorCompletionService =
   new ExecutorCompletionService<>>(mExecutorService);
```

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

```
mExecutorCompletionService =
   new ExecutorCompletionService<> (mExecutorService);
```

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

End of the Java Executor CompletionService: Introduction