Java ExecutorCompletionService: Introduction

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

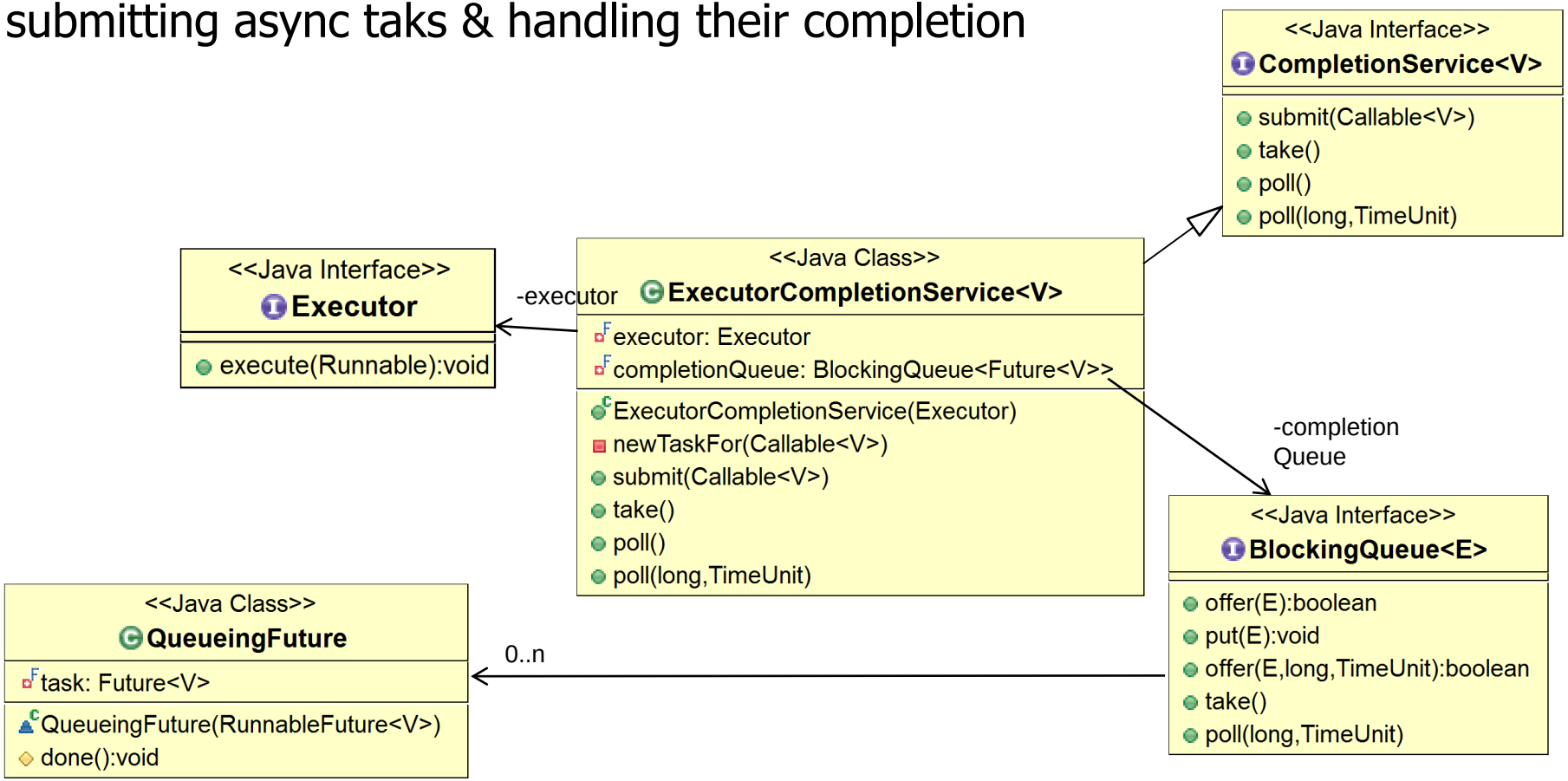
Professor of Computer Science

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 submitting async taks & handling their completion



Learning Objectives in this Part of the Lesson

- Understand how Java CompletionService's interface defines a framework for submitting async taks & handling their completion
- Know how to instantiate the Java ExecutorCompletionService

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

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

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

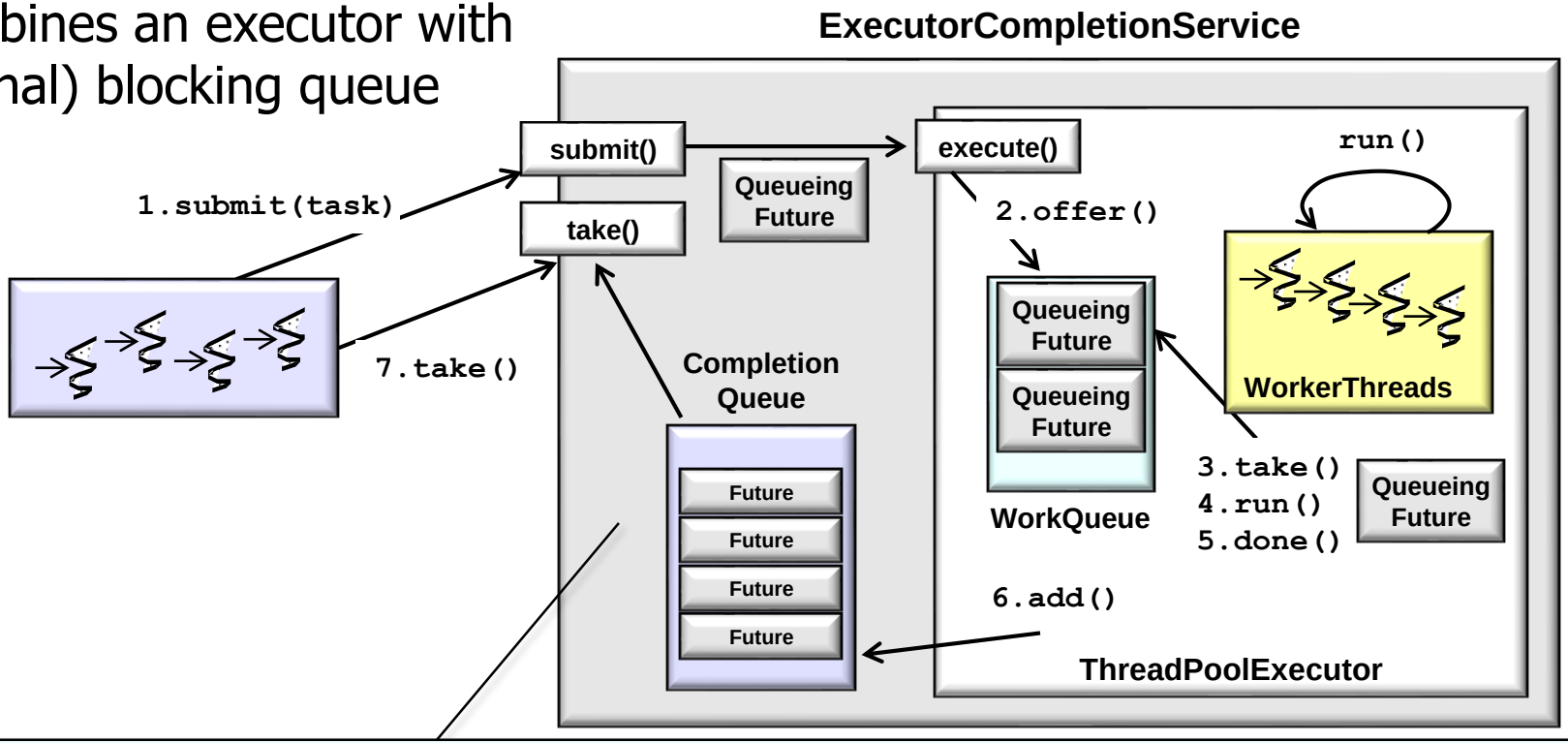
```
...
private class FutureRunnable
  implements Runnable {
  List<Future<PrimeCallable.PrimeResult>>
    mFutures;
  MainActivity mActivity; ...

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

future::get may block the thread, even if some other futures may have completed

This blocking problem is common w/the "synchronous future" processing model

Motivating the Java CompletionService Interface

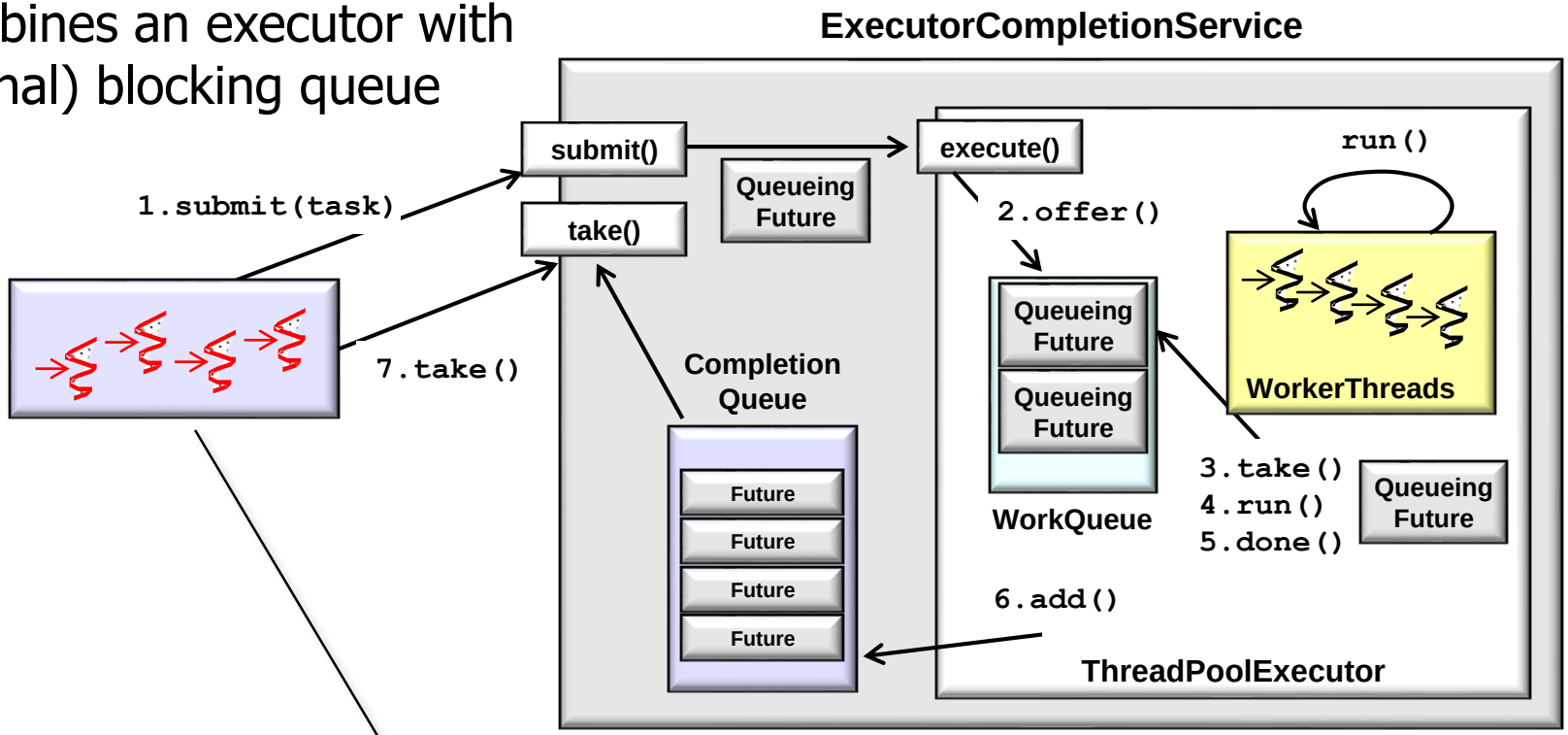
- CompletionService fixes this problem via an "async future" processing model that combines an executor with an (internal) blocking queue



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

Motivating the Java CompletionService Interface

- CompletionService fixes this problem via an "async future" processing model that combines an executor with an (internal) blocking queue

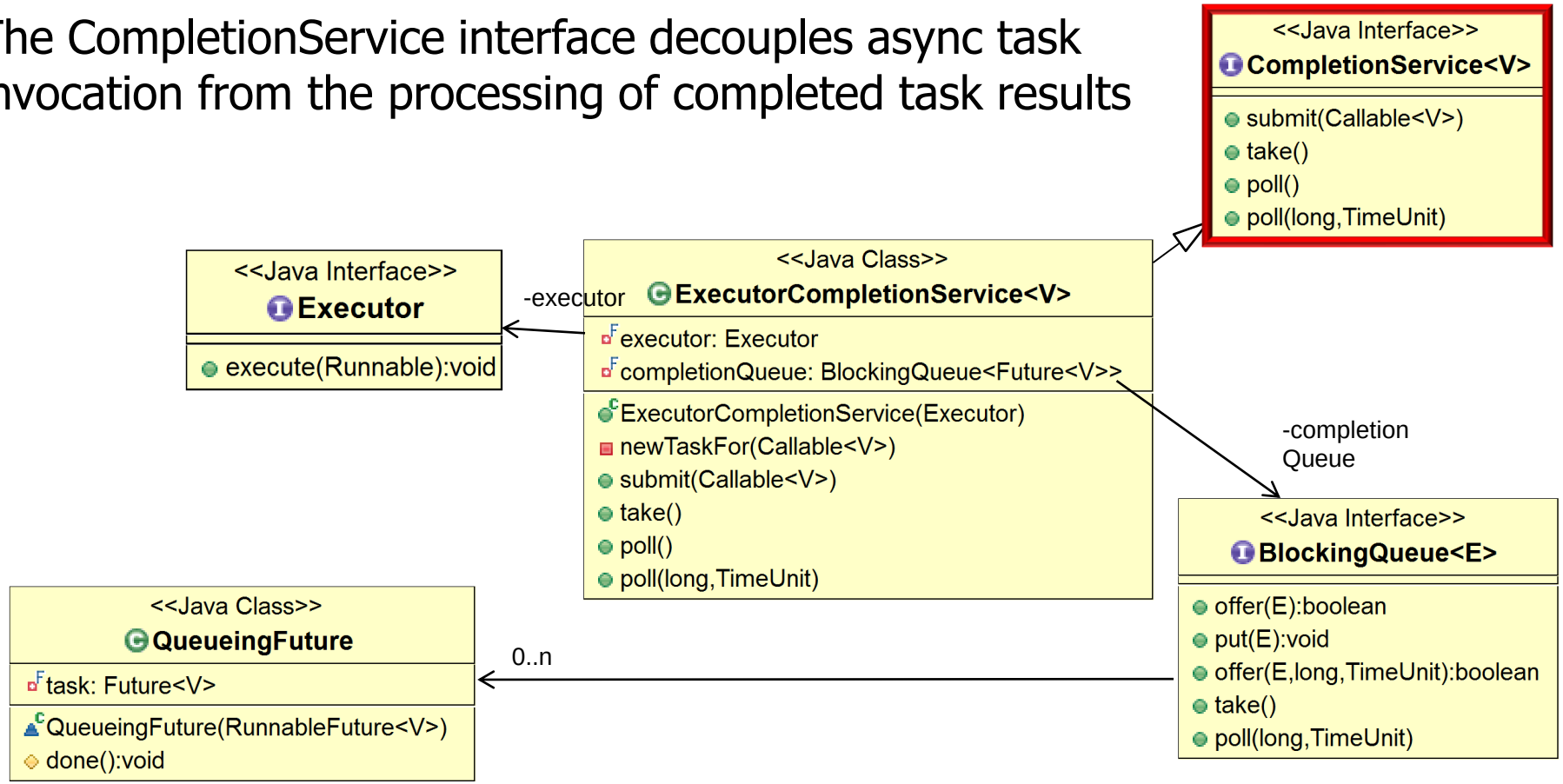


1+ client threads can submit tasks & 1+ client threads can process their results

Overview of the Java CompletionService Interface

Overview of the Java CompletionService Interface

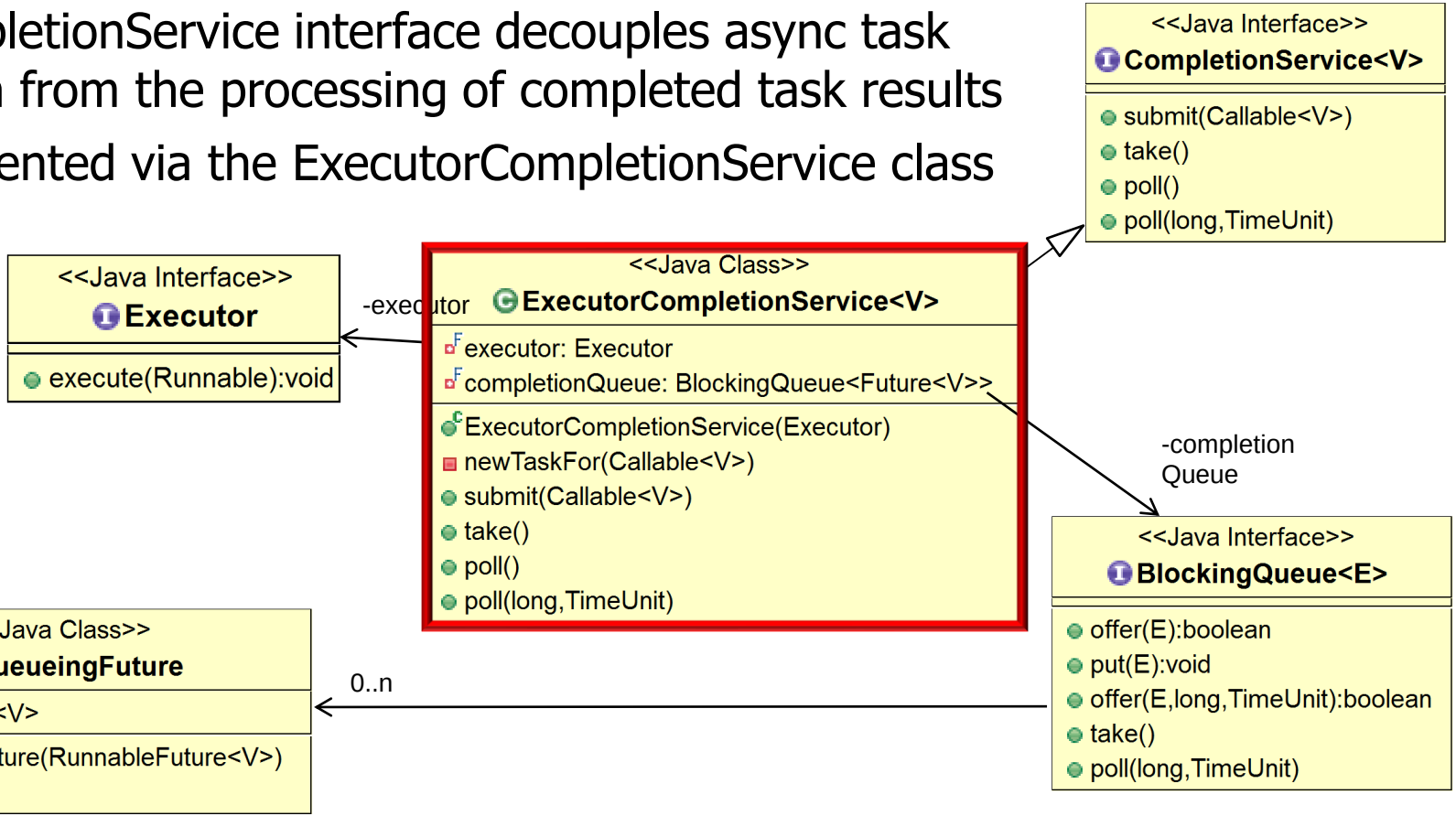
- The CompletionService interface decouples async task invocation from the processing of completed task results



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

Overview of the Java CompletionService Interface

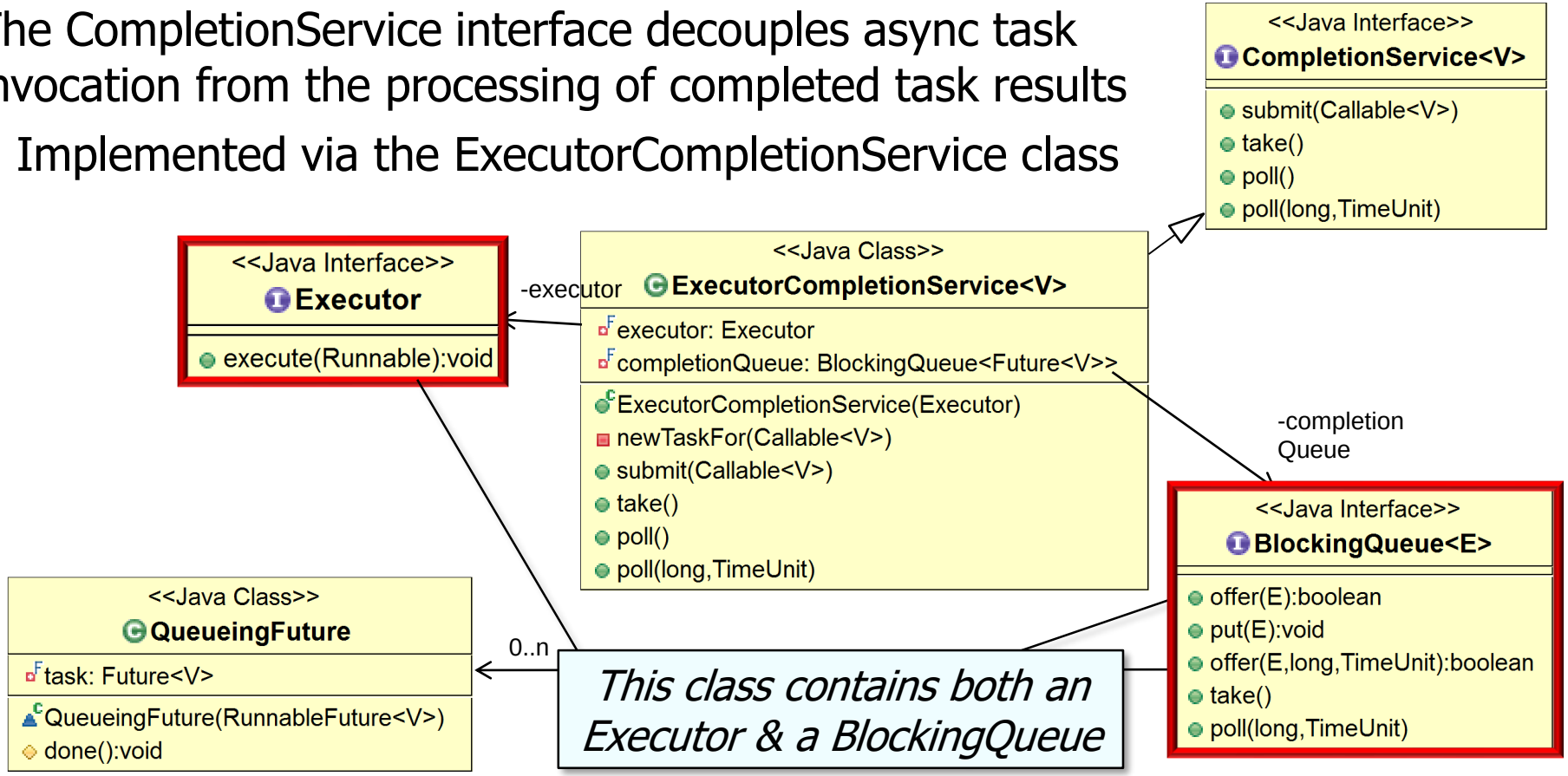
- The CompletionService interface decouples async task invocation from the processing of completed task results
  - Implemented via the ExecutorCompletionService class



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

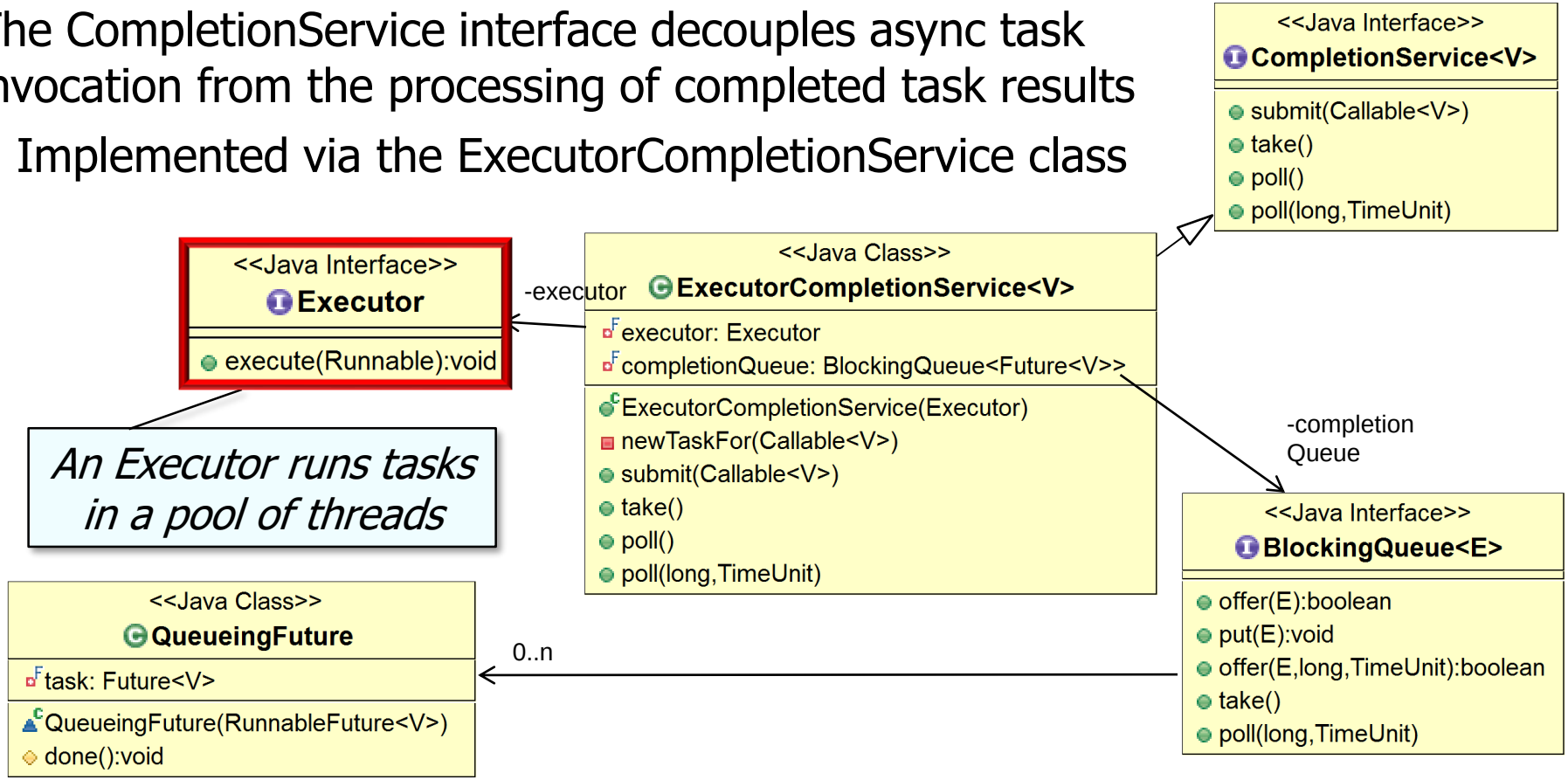
Overview of the Java CompletionService Interface

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



Overview of the Java CompletionService Interface

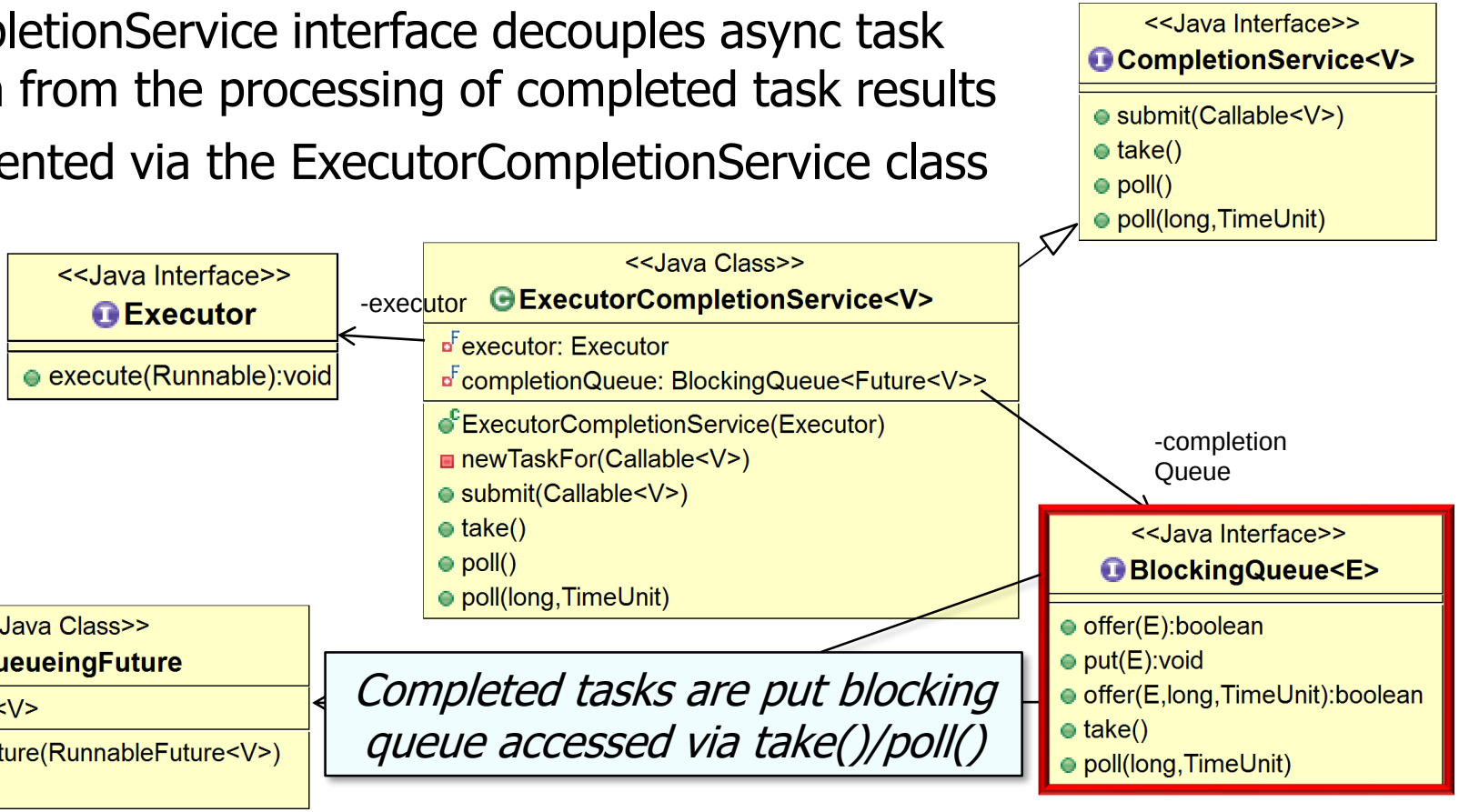
- The CompletionService interface decouples async task invocation from the processing of completed task results
  - Implemented via the ExecutorCompletionService class



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

Overview of the Java CompletionService Interface

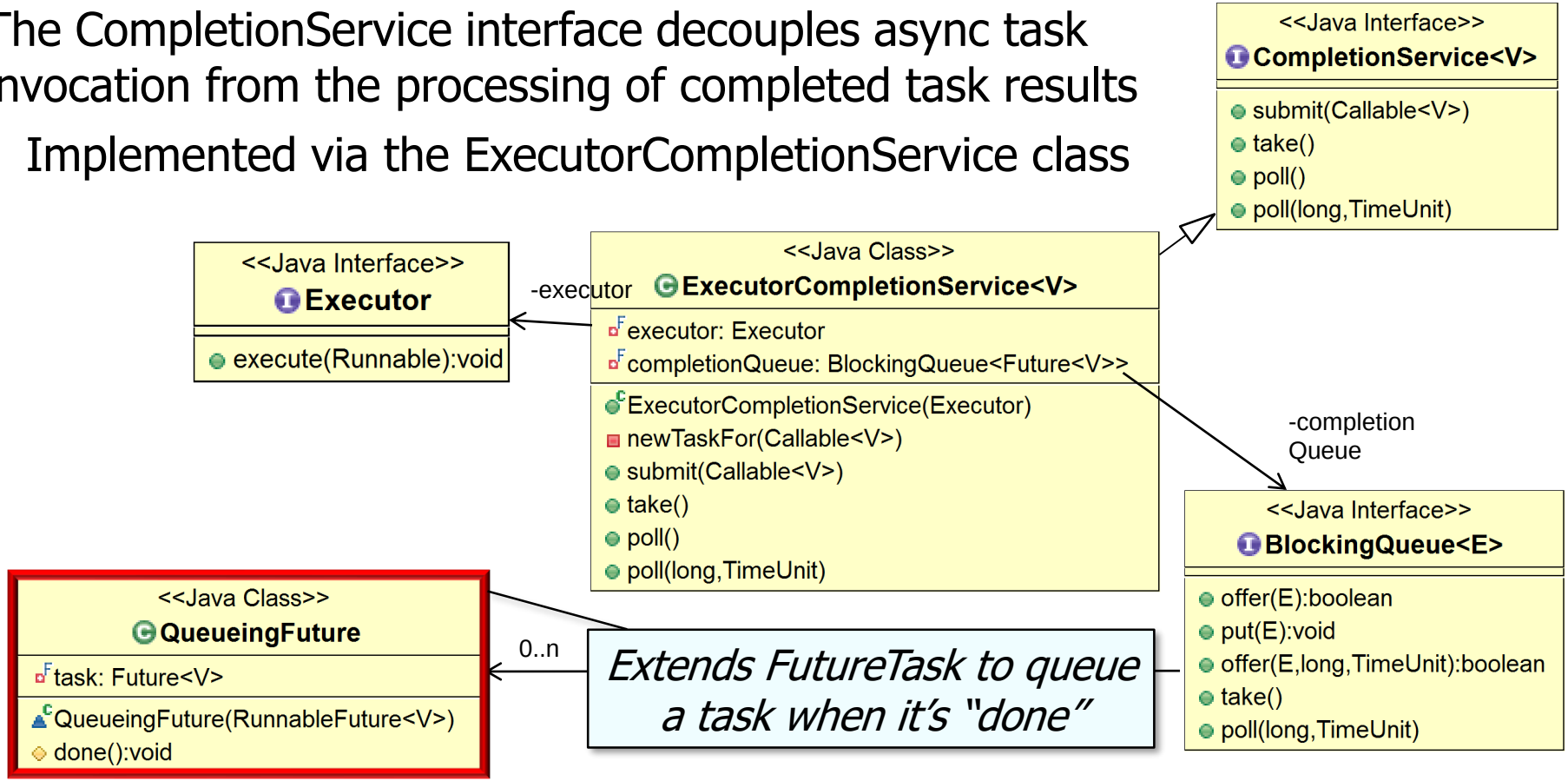
- The CompletionService interface decouples async task invocation from the processing of completed task results
  - Implemented via the ExecutorCompletionService class



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



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

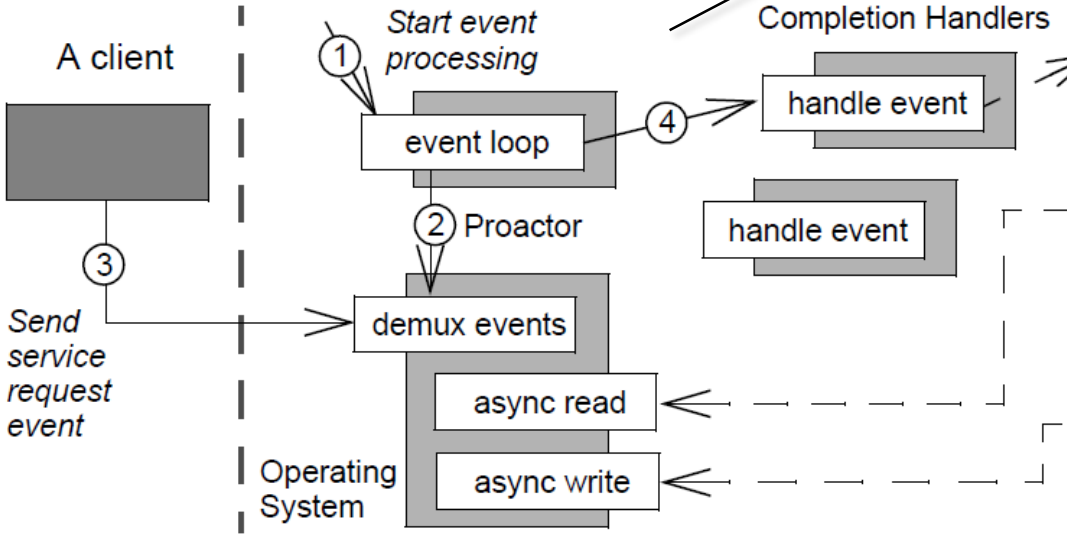
Overview of the Java CompletionService Interface

- CompletionService can implement the *Proactor* pattern

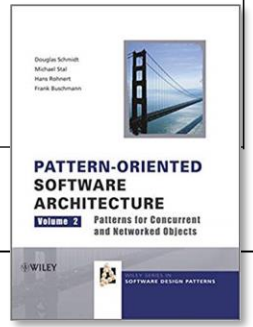
Supports demultiplexing & dispatching of event handlers that are triggered by the completion of async events

<<Java Interface>>
CompletionService<V>

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




```
handle_event (Event event)
begin ## Process the received event
  if (event.type == REQUEST)
     ## Read request asynchronously
     ## and return control.
     async_read ();

  elsif (event.type == READ_COMPLETE)
     ## Process event, deliver results
     ## asynchronously, and return control.
     process_data ();
     async_write ();
  fi
end
```

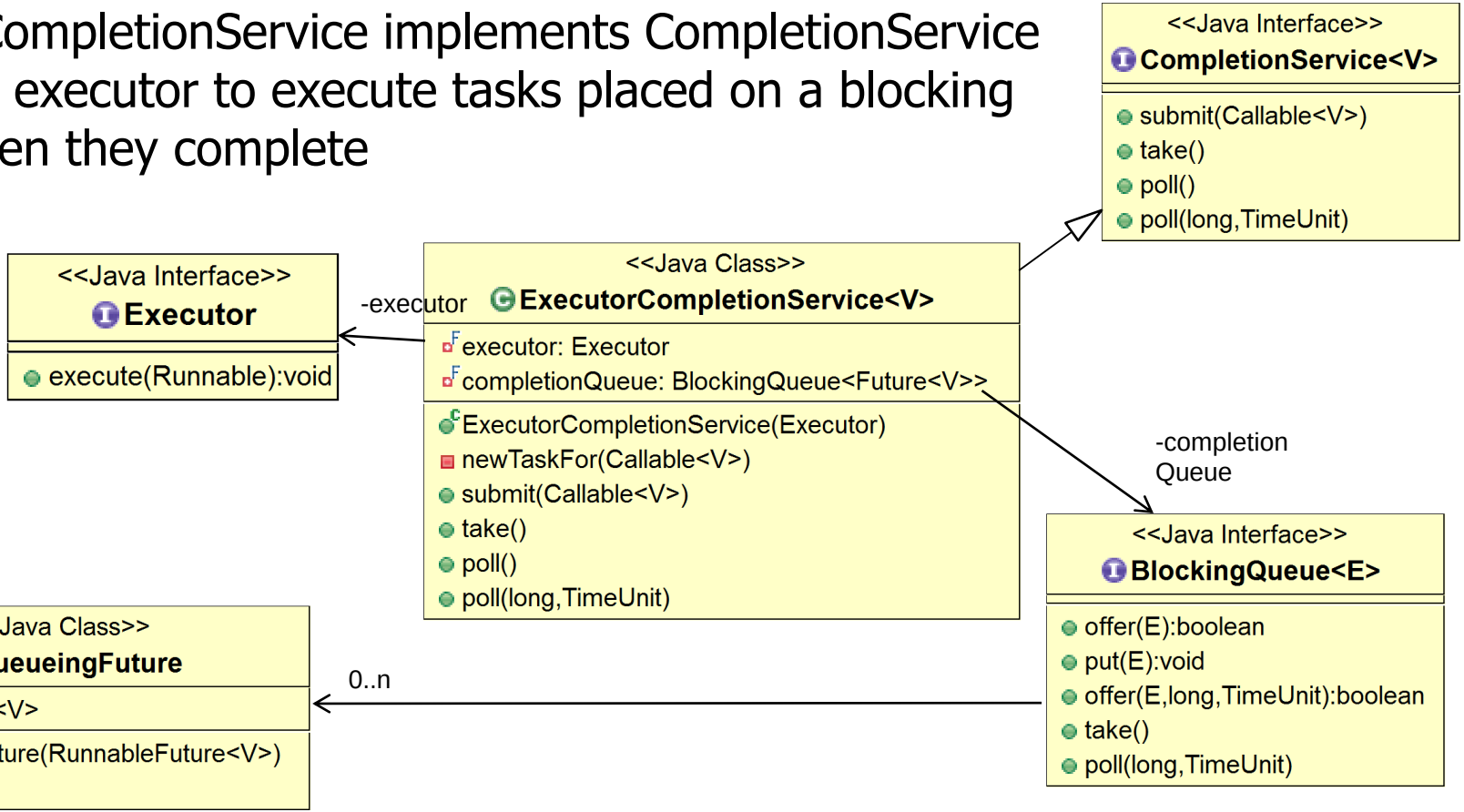


See en.wikipedia.org/wiki/Proactor_pattern

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

Instantiating the Java ExecutorCompletionService

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

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

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

Instantiating the Java ExecutorCompletionService

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

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

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

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

Instantiating the Java ExecutorCompletionService

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

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

Associate ExecutorCompletion Service with executor service

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

End of Java Executor CompletionService: Introduction