

Implementing the AsyncTaskBarrier Framework Using Project Reactor (Part 2)

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 the API of the `AsyncTaskBarrier` class
- Recognize how Project Reactor Flux & Mono operators are used to implement the `AsyncTaskBarrier` framework

```
static Mono<Long> runTests() {  
    ...  
    return Flux  
        .fromIterable(sTests)  
  
        .flatMap(Supplier::get)  
  
        .onErrorContinue(errorHandler)  
  
        .collectList()  
  
        .flatMap(__ -> Mono.just  
            (sTasks.size() -  
                exceptionCount  
                    .get()));  
}
```

See [Reactive/flux/ex4/src/main/java/Utils/AsyncTaskBarrier.java](https://github.com/reactor/reactor-examples/blob/master/reactor-examples-ex4/src/main/java/Utils/AsyncTaskBarrier.java)

Learning Objectives in this Part of the Lesson

- Understand the API of the `AsyncTaskBarrier` class
- Recognize how Project Reactor Flux & Mono operators are used to implement the `AsyncTaskBarrier` framework
- Recognize how the unit tests showcase the framework semantics & the subtleties between `onErrorContinue()`, `onErrorResume()`, & `onErrorStop()`

```
AsyncTaskBarrier
```

```
.register(throwException) ;
```

```
...
```

```
long testCount = AsyncTaskBarrier
```

```
// Run all the tests.
```

```
.runTasks()
```

```
// Block until all the tests
```

```
// are done to allow future
```

```
// computations to complete
```

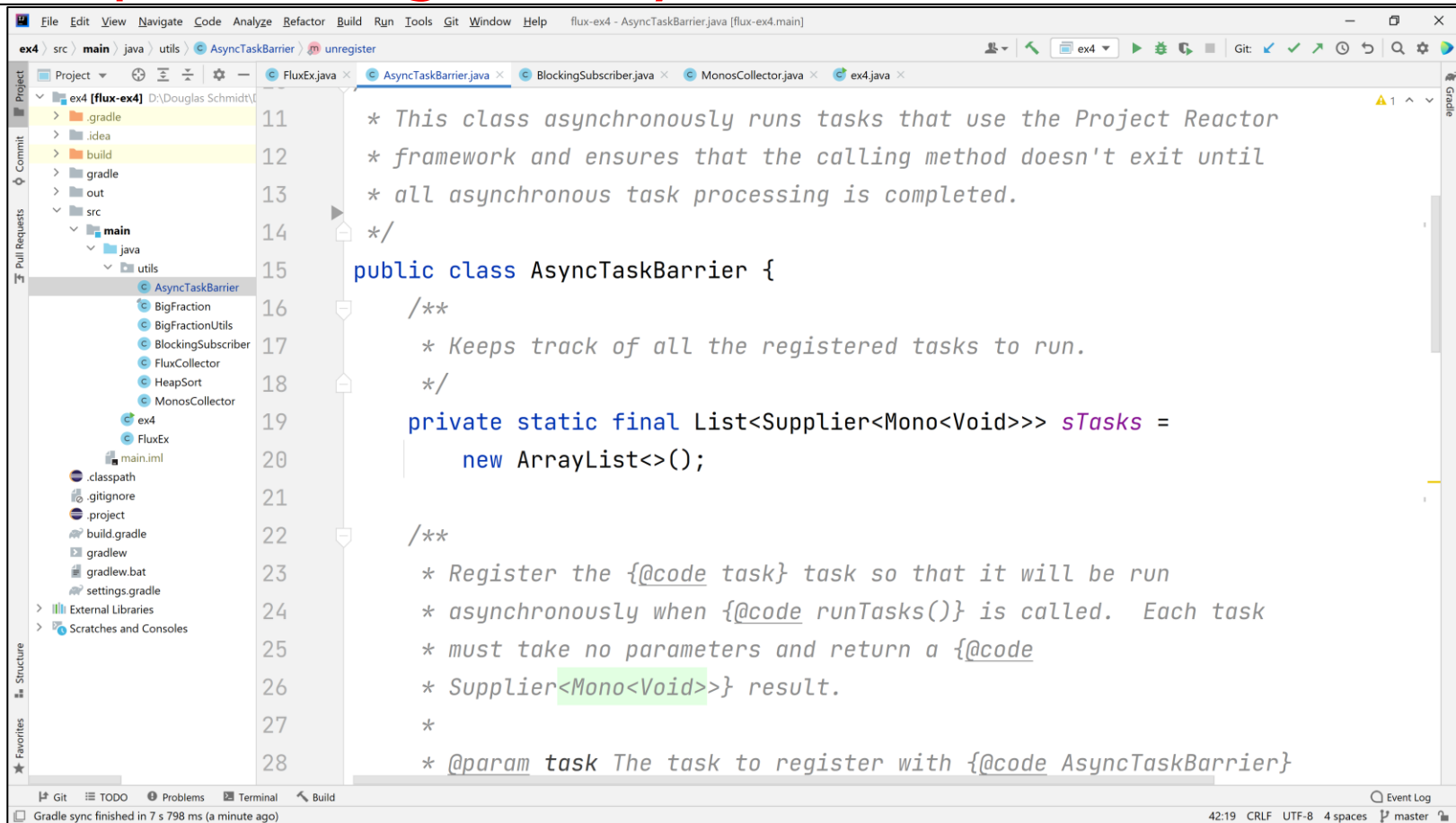
```
// running asynchronously.
```

```
.block();
```

```
assertEquals(testCount, 3);
```

Implementing the AsyncTaskBarrier Framework

Implementing the AsyncTaskBarrier Framework



The screenshot shows an IDE window with the following components:

- Project Explorer (Left):** Displays the project structure for 'ex4 [flux-ex4]'. The 'src/main/java/Utils' package is expanded, showing the 'AsyncTaskBarrier' class selected.
- Editor (Center):** Shows the source code of 'AsyncTaskBarrier.java'. The code includes a class-level comment, a class declaration, a static final list of tasks, and a comment for a method that is not yet implemented.
- Bottom Panel:** Includes a 'Build' tab showing 'Gradle sync finished in 7 s 798 ms (a minute ago)' and an 'Event Log'.

```
11  * This class asynchronously runs tasks that use the Project Reactor
12  * framework and ensures that the calling method doesn't exit until
13  * all asynchronous task processing is completed.
14  */
15  public class AsyncTaskBarrier {
16      /**
17       * Keeps track of all the registered tasks to run.
18       */
19      private static final List<Supplier<Mono<Void>>> sTasks =
20          new ArrayList<>();
21
22      /**
23       * Register the {@code task} task so that it will be run
24       * asynchronously when {@code runTasks()} is called. Each task
25       * must take no parameters and return a {@code
26       * Supplier<Mono<Void>>} result.
27       *
28       * @param task The task to register with {@code AsyncTaskBarrier}
```

See [Reactive/flux/ex4/src/main/java/Utils/AsyncTaskBarrier.java](https://github.com/reactive/flux-ex4/src/main/java/Utils/AsyncTaskBarrier.java)

Implementing the AsyncTaskBarrier Framework

- The register() & unregister() methods simply add & remove registered tasks to an internal list, respectively

```
static void register
    (Supplier<Mono<Void>> task){
    sTasks.add(task);
}
```

```
static boolean unregister
    (Supplier<Mono<Void>> task){
    return sTasks.remove(task);
}
```

Implementing the AsyncTaskBarrier Framework

- The register() & unregister() methods simply add & remove registered tasks to an internal list, respectively
- Each task is a Supplier whose get() method performs a task that returns Mono<Void>

```
static void register
    (Supplier<Mono<Void>> task) {
    sTasks.add(task);
}

static boolean unregister
    (Supplier<Mono<Void>> task) {
    return sTasks.remove(task);
}
```

Implementing the AsyncTaskBarrier Framework

- The register() & unregister() methods simply add & remove registered tasks to an internal list, respectively
- Each task is a Supplier whose get() method performs a task that returns Mono<Void>
 - This return type is used to signal when a task completes

```
static void register
    (Supplier<Mono<Void>> task) {
    sTasks.add(task);
}

static boolean unregister
    (Supplier<Mono<Void>> task) {
    return sTasks.remove(task);
}
```

Implementing the AsyncTaskBarrier Framework

- The runTasks() method runs all the registered tasks

```
static Mono<Long> runTests() {  
    ...  
    return Flux  
        .fromIterable(sTests)  
  
        .flatMap(Supplier::get)  
  
        .onErrorContinue(errorHandler)  
  
        .collectList()  
  
        .flatMap(__ -> Mono.just  
            (sTasks.size() -  
            exceptionCount  
            .get()));  
}
```

Implementing the AsyncTaskBarrier Framework

- The runTasks() method runs all the registered tasks
- It returns a Mono<Long> that triggers when all tasks complete

Emits the # of tasks that completed successfully

```
static Mono<Long> runTests() {  
    ...  
    return Flux  
        .fromIterable(sTests)  
        .flatMap(Supplier::get)  
        .onErrorContinue(errorHandler)  
        .collectList()  
  
        .flatMap(__ -> Mono.just  
            (sTasks.size() -  
             exceptionCount  
              .get()));  
}
```

Implementing the AsyncTaskBarrier Framework

- The runTasks() method runs all the registered tasks
- It returns a Mono<Long> that triggers when all tasks complete

Factory method that converts the list of suppliers into a Flux stream of suppliers

```
static Mono<Long> runTests() {  
    ...  
    return Flux  
        .fromIterable(sTests)  
  
        .flatMap(Supplier::get)  
  
        .onErrorContinue(errorHandler)  
  
        .collectList()  
  
        .flatMap(__ -> Mono.just  
            (sTasks.size() -  
                exceptionCount  
                .get()));  
}
```

Implementing the AsyncTaskBarrier Framework

- The runTasks() method runs all the registered tasks
- It returns a Mono<Long> that triggers when all tasks complete

```
static Mono<Long> runTests() {  
    ...  
    return Flux  
        .fromIterable(sTests)  
  
        .flatMap(Supplier::get)  
  
        .onErrorContinue(errorHandler)  
  
        .collectList()  
  
        .flatMap(__ -> Mono.just  
            (sTasks.size() -  
            exceptionCount  
            .get()));  
}
```

*Run all registered tasks, which
can execute asynchronously &
each return a Mono<Void>*

Implementing the AsyncTaskBarrier Framework

- The runTasks() method runs all the registered tasks
- It returns a Mono<Long> that triggers when all tasks complete

```
static Mono<Long> runTests() {  
    ...  
    return Flux  
        .fromIterable(sTests)  
  
        .flatMap(Supplier::get)  
  
        .onErrorContinue(errorHandler)  
  
        .collectList()  
  
        .flatMap(__ -> Mono.just  
            (sTasks.size() -  
             exceptionCount  
              .get()));  
}
```

*Log the exception &
continue processing*

Implementing the AsyncTaskBarrier Framework

- The runTasks() method runs all the registered tasks
- It returns a Mono<Long> that triggers when all tasks complete

```
static Mono<Long> runTests() {  
    ...  
    return Flux  
        .fromIterable(sTests)  
  
        .flatMap(Supplier::get)  
  
        .onErrorContinue(errorHandler)  
  
        .collectList()  
  
        .flatMap(__ -> Mono.just  
            (sTasks.size() -  
            exceptionCount  
            .get()));  
}
```

Collect into an empty list that triggers when all the tasks finish running asynchronously

Implementing the AsyncTaskBarrier Framework

- The runTasks() method runs all the registered tasks
- It returns a Mono<Long> that triggers when all tasks complete

```
static Mono<Long> runTests() {  
    ...  
    return Flux  
        .fromIterable(sTests)  
  
        .flatMap(Supplier::get)  
  
        .onErrorContinue(errorHandler)  
  
        .collectList()  
  
        .flatMap(__ -> Mono.just  
            (sTasks.size() -  
                exceptionCount  
                .get()));  
}
```

Return a mono containing the number of tasks run when all of the tasks have completed

End of Implementing the AsyncBarrierTask Framework Using Project Reactor (Part 2)