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

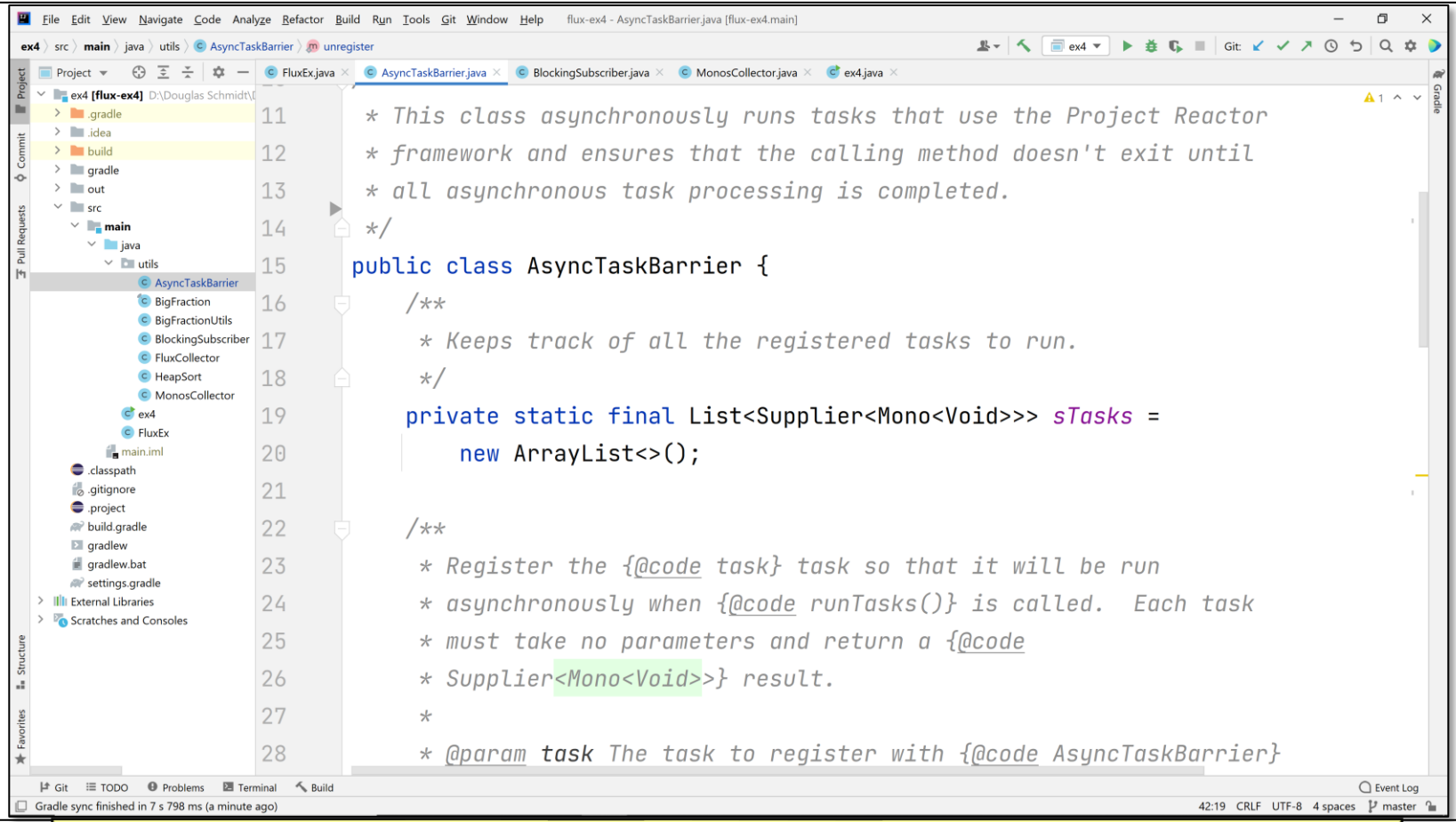
Learning Objectives in this Part of the Lesson

- 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 onError Continue(), 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



See [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

```
static Mono<Long> runTests() {
   ...
   return Flux
      .fromIterable(sTests)

      .flatMap(Supplier::get)

      .onErrorContinue(errorHandler)

      .collectList()

      .flatMap(__ -> Mono.just
                  (sTasks.size() -
                   exceptionCount
                     .get());
```

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

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#fromIterable

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>

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#flatMap

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

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#onErrorContinue

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

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#collectList

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

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#flatMap

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