Overview of the Project Reactor
AsyncTaskBarrier Framework

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 key classes in the Project Reactor API
• Understand key classes in the RxJava API
• Be aware of the structure & functionality of the BigFraction case studies
• Recognize the capabilities of the AsyncTaskBarrier framework for Project Reactor

There are implementations for both Project Reactor & RxJava
Overview of the Project
Reactor AsyncTask Barrier Class
Overview of the Project Reactor AsyncTaskBarrier Class

- Most test methods in the BigFraction case studies run asynchronously via subscribeOn(), so these methods return before their computations complete.

```java
global public static Mono<Void> testFractionReductionAsync() {
    BigFraction unreducedFraction = makeBigFraction(...);
    ...
    return Mono
        .fromCallable(() -> BigFraction.reduce(unreducedFraction))
        .subscribeOn(Schedulers.single())
        .map(result -> result.toMixedString())
        .doOnSuccess(result ->
            System.out.println
                ("big fraction = "
                    + result + "\n"))
        .then();
```

See Reactive/Mono/ex2/src/main/java/MonoEx.java
Overview of the Project Reactor AsyncTaskBarrier Class

- It’s therefore helpful to define a single location in the main driver program that waits for all asynchronously executing test methods to complete.

```java
public static void main (String[] argv) ... {
    AsyncTaskBarrier
        .register(MonoEx::testFractionReductionAsync);
    AsyncTaskBarrier
        .register(MonoEx::testFractionMultiplicationCallable1);
    AsyncTaskBarrier
        .register(MonoEx::testFractionMultiplicationCallable2);

    long testCount = AsyncTaskBarrier
        .runTasks()
        .block();

    ...
}
```

See Reactive/Mono/ex2/src/main/java/ex2.java
Overview of the Project Reactor AsyncTaskBarrier Class

- The AsyncTaskBarrier class provides an API to register non-blocking task methods that run *asynchronously*

```java
public static void main (String[] argv) {
    AsyncTaskBarrier
        .register(MonoEx::testFractionReductionAsync);
    AsyncTaskBarrier
        .register(MonoEx::testFractionMultiplicationCallable1);
    AsyncTaskBarrier
        .register(MonoEx::testFractionMultiplicationCallable2);

    long testCount = AsyncTaskBarrier
        .runTasks()
        .block();
    ...
}
```

*We use the registered task methods to run async tests*
Overview of the Project Reactor AsyncTaskBarrier Class

- The AsyncTaskBarrier class provides an API to register non-blocking task methods that run asynchronously

```java
public static void main (String[] argv) ... {
    AsyncTaskBarrier
        .register(MonoEx::testFractionReductionAsync);
    AsyncTaskBarrier
        .register(MonoEx::testFractionMultiplicationCallable1);
    AsyncTaskBarrier
        .register(MonoEx::testFractionMultiplicationCallable2);

    long testCount = AsyncTaskBarrier
        .runTasks()
        .block();
    ...
}
```

This framework also handles task methods that run and/or block synchronously
Overview of the Project Reactor AsyncTaskBarrier Class

• All of the registered task methods start running (a)synchronously when
AsyncTaskBarrier.runTasks() is called

```
public static void main(String[] argv) {
    AsyncTaskBarrier
        .register(MonoEx::testFractionReductionAsync);
    AsyncTaskBarrier
        .register(MonoEx::testFractionMultiplicationCallable1);
    AsyncTaskBarrier
        .register(MonoEx::testFractionMultiplicationCallable2);

    long testCount = AsyncTaskBarrier
        .runTasks()
        .block();
    ...
}
```
Overview of the Project Reactor AsyncTaskBarrier Class

- The driver program then calls block() on the Mono returned from runTasks() to wait for all asynchronous task processing to complete

```java
public static void main (String[] argv) ... {
    AsyncTaskBarrier
    .register(MonoEx::testFractionReductionAsync);
    AsyncTaskBarrier
    .register(MonoEx::testFractionMultiplicationCallable1);
    AsyncTaskBarrier
    .register(MonoEx::testFractionMultiplicationCallable2);

    long testCount = AsyncTaskBarrier
    .runTasks()
    .block();
    ...
}
```

Plays the role of a barrier synchronizer

See [en.wikipedia.org/wiki/Barrier_(computer_science)](en.wikipedia.org/wiki/Barrier_(computer_science))
Overview of the Project Reactor AsyncTaskBarrier Class

- The AsyncTaskBarrier framework (a)synchronously registers/runs tasks & ensures the calling method doesn’t exit until all async processing completes.

Class AsyncTaskBarrier

```java
public class AsyncTaskBarrier
extends java.lang.Object

This class asynchronously runs tasks that use the Project Reactor framework and ensures that the calling method doesn’t exit until all asynchronous task processing is completed.
```

### Method Summary

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void</td>
<td>register( java.util.function.Supplier&lt;reactor.core.publisher.Mono&lt;java.lang.Void&gt;&gt; task)</td>
<td>Register the task so that it will be run asynchronously when runTasks() is called.</td>
</tr>
<tr>
<td>static</td>
<td>runTasks()</td>
<td>Run all the register tasks.</td>
</tr>
</tbody>
</table>

See Reactive/mono/ex1/src/main/java/utils/AsyncTaskBarrier.java
Overview of the Project Reactor AsyncTaskBarrier Class

• The AsyncTaskBarrier framework (a)synchronously registers/runs tasks & ensures the calling method doesn’t exit until all async processing completes

Class AsyncTaskBarrier

public class AsyncTaskBarrier
extends java.lang.Object

This class asynchronously runs tasks that use the Project Reactor framework and ensures that the calling method doesn’t exit until all asynchronous task processing is completed.

Method Summary

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void</td>
<td>register(java.util.function.Supplier&lt;reactor.core.publisher.Mono&lt;java.lang.Void&gt;&gt; task)</td>
<td>Register the task task so that it will be run asynchronously when runTasks() is called.</td>
</tr>
<tr>
<td>static</td>
<td>runTasks()</td>
<td>Run all the register tasks.</td>
</tr>
</tbody>
</table>
End of Overview of the Project Reactor Async TaskBarrier Framework