Overview of the Project Reactor AsyncTaskBarrier Framework

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

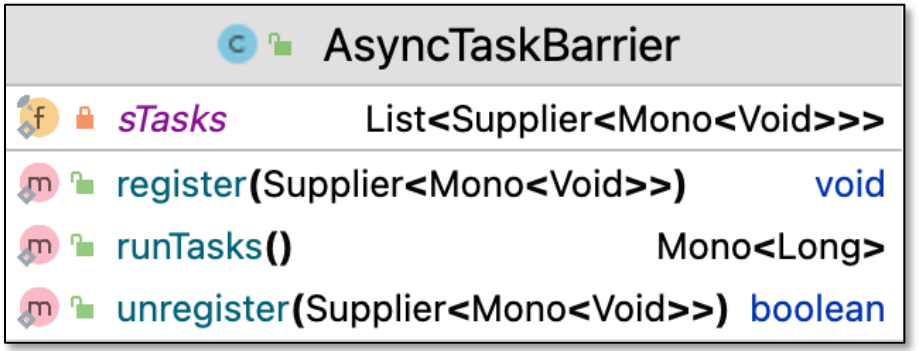
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand key classes in the Project Reactor API
- Know how Project Reactor Flux supports backpressure
- Recognize the Flux overflow strategies
- Be aware of the structure & functionality of the BigFraction case studies



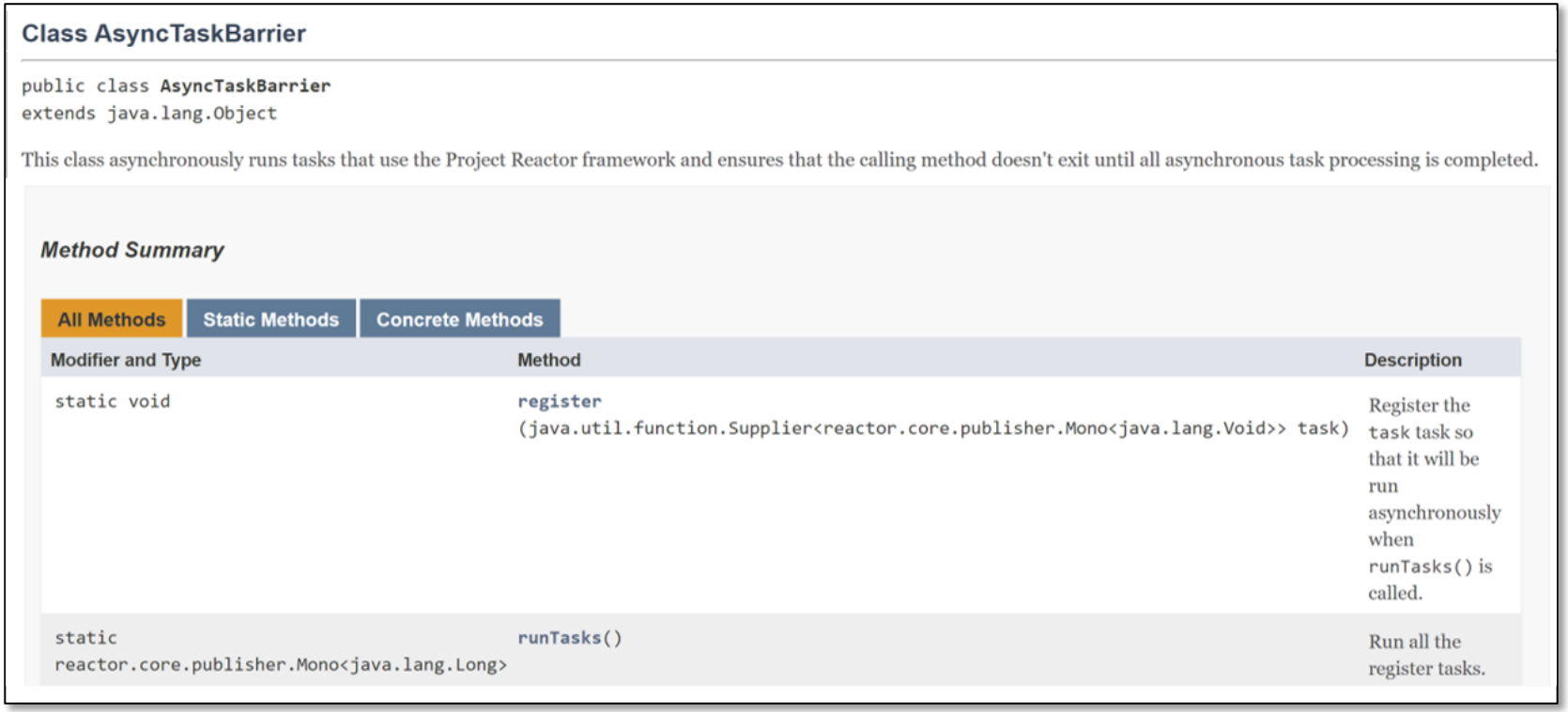
- Recognize the capabilities of the AsyncTaskBarrier framework for Project Reactor
  - This framework provides an API to register (non-)blocking task methods that run *(a)synchronously*

There are implementations for both Project Reactor & RxJava

Overview of the Project Reactor AsyncTaskBarrier Class

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

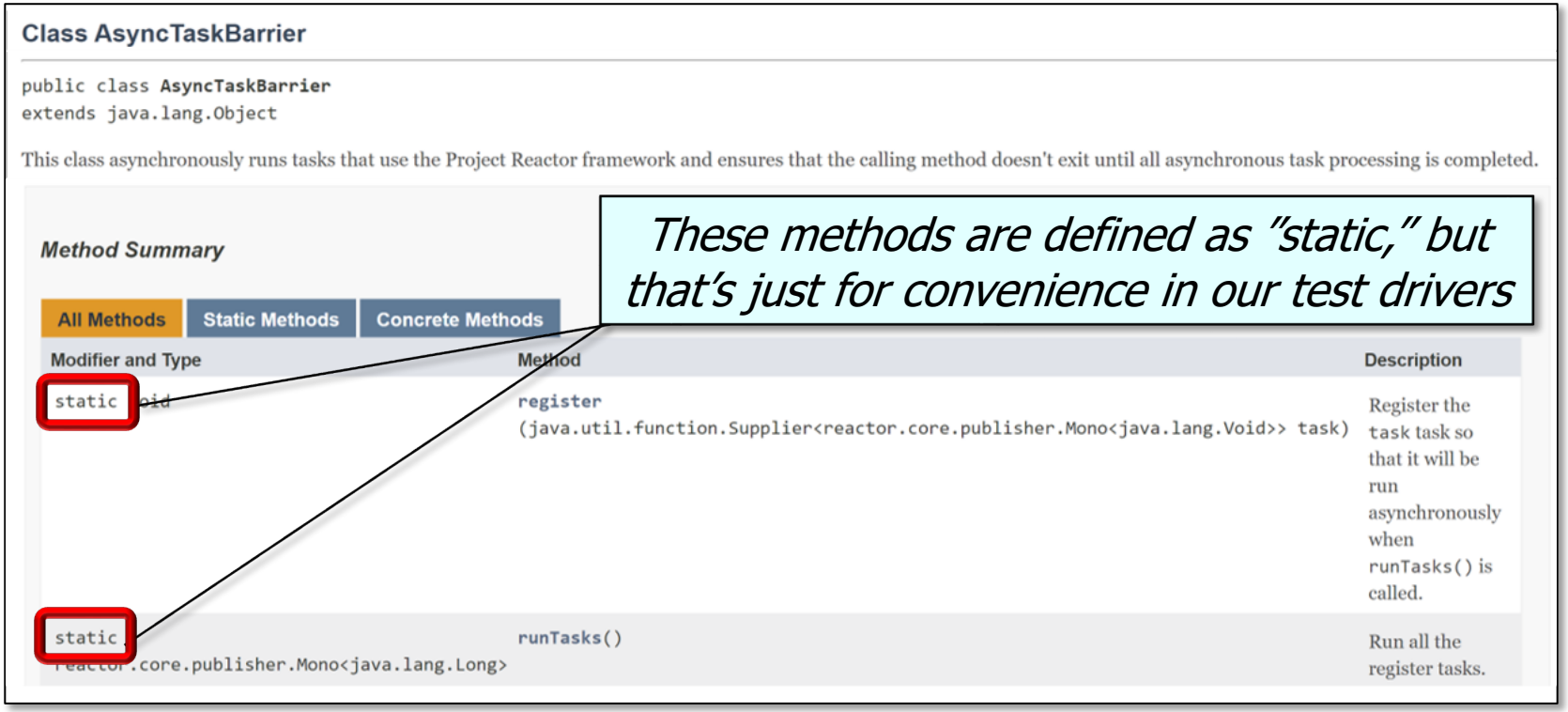
All Methods | Static Methods | Concrete Methods

Modifier and Type	Method	Description
static void	register (java.util.function.Supplier<reactor.core.publisher.Mono<java.lang.Void>> task)	Register the task task so that it will be run asynchronously when runTasks() is called.
static reactor.core.publisher.Mono<java.lang.Long>	runTasks()	Run all the register tasks.

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.

These methods are defined as "static," but that's just for convenience in our test drivers

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method	Description
static void	register (java.util.function.Supplier<reactor.core.publisher.Mono<java.lang.Void>> task)	Register the task task so that it will be run asynchronously when runTasks() is called.
static reactor.core.publisher.Mono<java.lang.Long>	runTasks()	Run all the register tasks.

See www.baeldung.com/java-static-methods-use-cases

Overview of the Project Reactor AsyncTaskBarrier Class

- Most test methods in the BigFraction case studies run async via subscribeOn()

```
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

- Most test methods in the BigFraction case studies run async via subscribeOn()
  - These methods can therefore return before their computations complete

```
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();
```

Async methods cause chaos & insanity in some circumstances

See www.upgrad.com/tutorials/software-engineering/java-tutorial/daemon-thread-in-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

```
public static void main (String[] argv) ... {
  AsyncTaskBarrier
    .register(FluxEx::testIsPrimeTimed);
  AsyncTaskBarrier
    .register(FluxEx::testIsPrimeAsync);

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

Overview of the Project Reactor AsyncTaskBarrier Class

- The AsyncTaskBarrier class provides an API to register (non-)blocking task methods that run *(a)synchronously*

```
public static void main (String[] argv) ... {
  AsyncTaskBarrier
    .register(FluxEx::testIsPrimeTimed);
  AsyncTaskBarrier
    .register(FluxEx::testIsPrimeAsync);

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

We use the registered task methods to run async tests

See Reactive/flux/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 *(a)synchronously*

```
public static void main (String[] argv) ... {
  AsyncTaskBarrier
    .register(FluxEx::testFractionMultiplicationSync1);
  AsyncTaskBarrier
    .register(FluxEx::testFractionMultiplicationSync2);
  ...

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

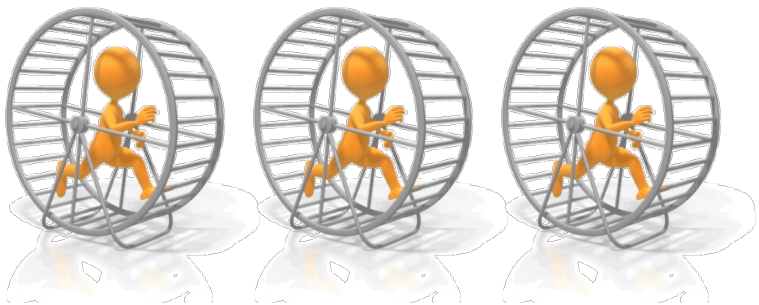
This framework also handles task methods that run and/or block synchronously

See Reactive/flux/ex1/src/main/java/ex1.java

Overview of the Project Reactor AsyncTaskBarrier Class

- All registered task methods run (a)synchronously when AsyncTaskBarrier .runTasks() is called

```
public static void main (String[] argv) ... {
  AsyncTaskBarrier
    .register(FluxEx::testIsPrimeTimed);
  AsyncTaskBarrier
    .register(FluxEx::testIsPrimeAsync);

  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

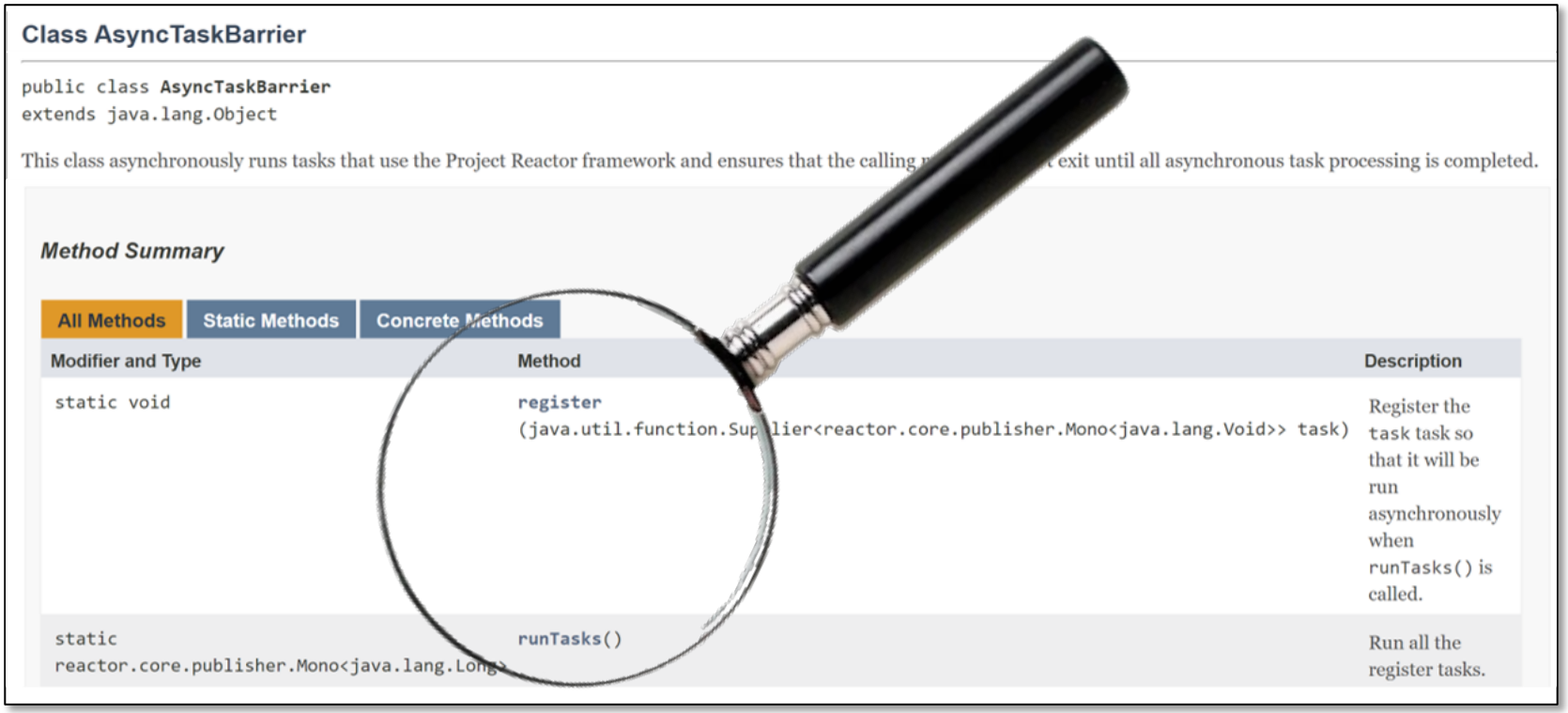
```
public static void main (String[] argv) ... {
   AsyncTaskBarrier
      .register(FluxEx::testIsPrimeTimed);
   AsyncTaskBarrier
      .register(FluxEx::testIsPrimeAsync);

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

Plays the role of a barrier synchronizer

See en.wikipedia.org/wiki/Barrier_(computer_science)

Overview of the Project Reactor AsyncTaskBarrier Class

- We'll explore the AsyncTaskBarrier framework's implementation after covering the Project Reactor Flux operators in more detail



See Reactive/flux/ex1/src/main/java/utils/AsyncTaskBarrier.java

End of Overview of the Project Reactor Async TaskBarrier Framework