Learning Objectives in this Part of the Lesson

• Recognize key Mono operators
• Concurrency & scheduler operators
  • These operators arrange to run other operators in designated threads & thread pools
  • e.g., Mono.subscribeOn() & Schedulers.single()
Key Concurrency Operators in the Mono Class
Key Concurrency Operators in the Mono Class

- The `subscribeOn()` operator
  - Run `subscribe()`, `onSubscribe()`, & `request()` on the specified Scheduler worker

```java
Mono<T> subscribeOn(Scheduler scheduler)
```
Key Concurrency Operators in the Mono Class

- The subscribeOn() operator
  - Run subscribe(), onSubscribe(), & request() on the specified Scheduler worker
  - The scheduler param indicates what thread to perform the operation on

```java
Mono<T> subscribeOn(Scheduler scheduler)
```

See [projectreactor.io/docs/core/release/api/reactor/core/scheduler/Scheduler.html](projectreactor.io/docs/core/release/api/reactor/core/scheduler/Scheduler.html)
• The subscribeOn() operator
  • Run subscribe(), onSubscribe(), & request() on the specified Scheduler worker
    • The scheduler param indicates what thread to perform the operation on
  • Returns the Mono requesting async processing
Key Concurrency Operators in the Mono Class

• The subscribeOn() operator
  • Run subscribe(), onSubscribe(), & request() on the specified Scheduler worker
• The semantics of subscribeOn() are a bit unusual
The `subscribeOn()` operator
- Run `subscribe()`, `onSubscribe()`, & `request()` on the specified Scheduler worker
- The semantics of `subscribeOn()` are a bit unusual
- Placing this operator in a chain impacts the execution context of the `onNext()`, `onError()`, & `onComplete()` signals

Run all this processing in a background thread

```java
return Mono
    .fromCallable(() -> BigFraction
        .reduce(unreducedFrac))
    .subscribeOn(Schedulers.single())
    .doOnSuccess(bf -> logBigFraction
        (unreducedFrac, bf, sb))
    .map(BigFraction::toMixedString)
    .doOnSuccess(bf ->
        displayBigFraction(bf, sb))
    .then();
```

See Reactive/mono/ex2/src/main/java/MonoEx.java
Key Concurrency Operators in the Mono Class

- The `subscribeOn()` operator
- Run `subscribe()`, `onSubscribe()`, & `request()` on the specified Scheduler worker
- The semantics of `subscribeOn()` are a bit unusual
- Placing this operator in a chain impacts the execution context of the `onNext()`, `onError()`, & `onComplete()` signals

```
return Mono
  .fromCallable(() -> BigFraction
    .reduce(unreducedFrac)
      .doOnSuccess(bf -> logBigFraction
        (unreducedFrac, bf, sb))
      .map(BigFraction::toMixedString)
      .doOnSuccess(bf ->
        displayBigFraction(bf, sb))
    .subscribeOn(Schedulers.single())
  .then();
```

`subscribeOn()` can appear later in the chain & have the same effect
Key Concurrency Operators in the Mono Class

- The subscribeOn() operator
  - Run subscribe(), onSubscribe(), & request() on the specified Scheduler worker

- The semantics of subscribeOn() are a bit unusual
  - Placing this operator in a chain impacts the execution context of the onNext(), onError(), & onComplete() signals
  - However, if a publishOn() operator appears later in the chain that will change the threading context where the rest of the operators in the chain below it execute (publishOn() can appear multiple times)

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#publishOn](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#publishOn)
Key Concurrency Operators in the Mono Class

• The `subscribeOn()` operator
  • Run `subscribe()`, `onSubscribe()`, & `request()` on the specified Scheduler worker
  • The semantics of `subscribeOn()` are a bit unusual
  • RxJava’s `Single.subscribeOn()` works the same way

Key Scheduler

Operators in the Mono Class
Key Scheduler Operators in the Mono Class

- The Schedulers.single() operator
  - Hosts a single-threaded Executor Service-based worker that runs concurrently wrt the caller

See [projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#single](http://projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#single)
Key Scheduler Operators in the Mono Class

- The Schedulers.single() operator

  ```java
  static Scheduler single()
  ```

- Hosts a single-threaded Executor Service-based worker that runs concurrently wrt the caller

  ```java
  return Mono
    .fromCallable(() -> BigFraction.reduce(unreducedFrac))
    .subscribeOn(Schedulers.single())
    .doOnSuccess(bf -> logBigFraction(unreducedFrac, bf, sb))
    .map(BigFraction::toMixedString)
    .doOnSuccess(bf -> displayBigFraction(bf, sb))
    .then();
  ```

  Run all this processing in a single background thread

See Reactive/mono/ex2/src/main/java/MonoEx.java
The Schedulers.single() operator

- Hosts a single-threaded Executor Service-based worker that runs concurrently wrt the caller
- Optimized for low-latency calls that all run in one (& only one) background thread

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html
Key Scheduler Operators in the Mono Class

- The Schedulers.single() operator
- Hosts a single-threaded Executor Service-based worker that runs concurrently wrt the caller
  - Optimized for low-latency calls that all run in one (& only one) background thread
- Implemented via a “daemon thread”
  - i.e., won’t prevent the app from exiting even if its work isn’t done

See [www.baeldung.com/java-daemon-thread](http://www.baeldung.com/java-daemon-thread)
Key Scheduler Operators in the Mono Class

- The Schedulers.single() operator
  - Hosts a single-threaded Executor Service-based worker that runs concurrently wrt the caller
- RxJava’s Schedulers.single() works the same way

```java
@NonNull
public static @NonNull Scheduler single()

Returns a default, shared, single-thread-backed Scheduler instance for work requiring strongly-sequential execution on the same background thread.

Uses:
- event loop
- support Schedulers.from(Executor) and from(ExecutorService) with delayed scheduling
- support benchmarks that pipeline data from one thread to another thread and avoid core-bashing of computation's round-robin nature

Unhandled errors will be delivered to the scheduler Thread's Thread.UncaughtExceptionHandler.
```

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/schedulers/Schedulers.html#single](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/schedulers/Schedulers.html#single)
End of Key Concurrency & Scheduler Operators in the Mono Class (Part 1)