Key Blocking Operators in the Mono Class

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

- Recognize key Mono operators
  - Concurrency & scheduler operators
- Blocking operators
  - These operators block awaiting a Mono to emit its value
    - e.g., block() & blockOptional()

The Mono that emits a value typically runs asynchronously in a different thread of control.
Key Blocking Operators in the Mono Class
Key Blocking Operators in the Mono Class

- The block() operator
- Subscribe to this Mono & block until a next signal is received

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#block
The block() operator

Subscribe to this Mono & block until a next signal is received

- Returns the value received or null if the Mono completes empty
Key Blocking Operators in the Mono Class

- The `block()` operator
  - Subscribe to this Mono & block until a next signal is received
    - Returns the value received or null if the Mono completes empty
    - If the Mono errors, the original exception is thrown
The block() operator

Subscribe to this Mono & block until a next signal is received

- Returns the value received or null if the Mono completes empty

- If the Mono errors, the original exception is thrown

- A checked exception is wrapped in a RuntimeException

See [docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html](docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html)
Key Blocking Operators in the Mono Class

- The `block()` operator
  - Subscribe to this Mono & block until a next signal is received
    - Returns the value received or null if the Mono completes empty
    - If the Mono errors, the original exception is thrown
  - There's also a version of `block()` that blocks until a next signal is received or a timeout expires

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

- **The block() operator**
  - Subscribe to this Mono & block until a next signal is received
    - Returns the value received or null if the Mono completes empty
    - If the Mono errors, the original exception is thrown
  - There’s also a version of block() that blocks until a next signal is received or a timeout expires
    - If the provided timeout expires, a RuntimeException is thrown

See [docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html](http://docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html)
Key Blocking Operators in the Mono Class

- The `block()` operator
- Subscribe to this Mono & block until a next signal is received
  - Returns the value received or null if the Mono completes empty
  - If the Mono errors, the original exception is thrown
  - There’s also a version of `block()` that blocks until a next signal is received or a timeout expires
  - `block()` internally calls `subscribe()` to initiate the Mono processing chain

```java
T block(Duration timeout) {
    BlockingMonoSubscriber<T> subscriber = new BlockingMonoSubscriber<>();
    subscribe((Subscriber<T>) subscriber);
    return subscriber.blockingGet(
        timeout.toNanos(),
        TimeUnit.NANOSECONDS);
}
```

The block() operator

- Subscribe to this Mono & block until a next signal is received
- Should only be used if a value is needed before proceeding

BigFraction bf1 = ...
BigFraction bf2 = ...

BigFraction result = Mono
  .fromCallable(() -> bf1.multiply(bf2))
  .subscribeOn(Schedulers.single())
  .block();
System.out.println(result.toMixedString());
- The `block()` operator
  - Subscribe to this Mono & block until a next signal is received
  - Should only be used if a value is needed before proceeding
- RxJava’s `blockingGet()` operator is similar

```java
BigFraction bf1 = ...  
BigFraction bf2 = ...  
BigFraction result = Single
  .fromCallable(() -> bf1.multiply(bf2))  
  .subscribeOn(Schedulers.single())  
  .blockingGet();  
System.out.println(result.toMixedString());
```

Key Blocking Operators in the Mono Class

- The `block()` operator
  - Subscribe to this Mono & block until a next signal is received
  - Should only be used if a value is needed before proceeding
  - RxJava’s `blockingGet()` operator is similar
- Similar to `CompletableFuture.join()`

```java
CompletableFuture<BigFraction> f = CompletableFuture
  .supplyAsync(() -> {
    BigFraction bf1 = new BigFraction(sF1);
    BigFraction bf2 = new BigFraction(sF2);
    return bf1.multiply(bf2);
  });

... System.out.println("result = " + f.join().toMixedString());
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#join
Key Blocking Operators in the Mono Class

- The `blockOptional()` operator
- Subscribe to this Mono & block until a next signal is received or the Mono completes empty

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

- The `blockOptional()` operator
- Subscribe to this Mono & block until a next signal is received or the Mono completes empty
- Returns an Optional

See [docs.oracle.com/javase/8/docs/api/java/util/Optional.html](https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html)
Key Blocking Operators in the Mono Class

- The `blockOptional()` operator
  - Subscribe to this Mono & block until a next signal is received or the Mono completes empty
  - Returns an Optional
  - Can replace the empty case with an Exception via `Optional.orElseThrow()`

See [docs.oracle.com/javase/8/docs/api/java/util/Optional.html#orElseThrow](docs.oracle.com/javase/8/docs/api/java/util/Optional.html#orElseThrow)
Key Blocking Operators in the Mono Class

- The `blockOptional()` operator
  - Subscribe to this Mono & block until a next signal is received or the Mono completes empty
  - Returns an Optional
    - Can replace the empty case with an Exception via `Optional.orElseThrow()`
    - Can return a default value via `Optional.orElse()` or `orElseGet()`

See [docs.oracle.com/javase/8/docs/api/java/util/Optional.html#orElse](http://docs.oracle.com/javase/8/docs/api/java/util/Optional.html#orElse)
Key Blocking Operators in the Mono Class

- The blockOptional() operator
  - Subscribe to this Mono & block until a next signal is received or the Mono completes empty
  - Returns an Optional
  - Can replace the empty case with an Exception via Optional .orElseThrow()
  - Can return a default value via Optional.orElse() or orElseGet()
  - Eliminates the dreaded Java NullPointerException

See [www.amitph.com/avoid-nullpointerexception-using-java-8-optional](http://www.amitph.com/avoid-nullpointerexception-using-java-8-optional)
Key Blocking Operators in the Mono Class

- The blockOptional() operator
  - Subscribe to this Mono & block until a next signal is received or the Mono completes empty
    - Returns an Optional
    - There’s also a blockOptional() operator that blocks until a next signal is received or a timeout expires

Optional<T> blockOptional(Duration timeout)

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#blockOptional
Key Blocking Operators in the Mono Class

- The `blockOptional()` operator
  - Subscribe to this Mono & block until a next signal is received or the Mono completes empty
    - Returns an Optional
  - There’s also a `blockOptional()` operator that blocks until a next signal is received or a timeout expires
    - If the provided timeout expires, a `Runtime Exception` is thrown

See [docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html](docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html)
The `blockOptional()` operator

- Subscribe to this Mono & block until a next signal is received or the Mono completes empty
- Should only be used if a value is needed before proceeding

```java
BigFraction bf1 = ...  
BigFraction bf2 = ...  
Optional<BigFraction> result = Mono
    .fromCallable(() -> bf1.multiply(bf2))
    .subscribeOn(Schedulers.single())
    .blockOptional();
System.out.println(result.map(BigFraction::toMixedString)
    .orElse("error"));
```

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

• The blockOptional() operator
  • Subscribe to this Mono & block until a next signal is received or the Mono completes empty
  • Should only be used if a value is needed before proceeding
  • There’s no direct RxJava equivalent
The `blockOptional()` operator

- Subscribe to this Mono & block until a next signal is received or the Mono completes empty
- Should only be used if a value is needed before proceeding
- There’s no direct RxJava equivalent

```java
BigFraction bf1 = ...  
BigFraction bf2 = ...  
BigFraction result = Single
    .fromCallable(() -> bf1.multiply(bf2))
    .subscribeOn(Schedulers.single())
    .blockingGet();

System.out.println(result.toMixedString())
```

However, `blockingGet()` can wait for a result

Key Blocking Operators in the Mono Class

- The blockOptional() operator
  - Subscribe to this Mono & block until a next signal is received or the Mono completes empty
  - Should only be used if a value is needed before proceeding
  - There’s no direct RxJava equivalent
  - Similar to CompletableFuture.join()

```java
CompletableFuture<BigFraction> f = CompletableFuture.<BigFraction>supplyAsync(() -> {
    BigFraction bf1 = new BigFraction(sF1);
    BigFraction bf2 = new BigFraction(sF2);
    return bf1.multiply(bf2);
});
...
System.out.println("result = "+f.join().toMixedString());
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#join
End of Key Blocking Operators in the Mono Class