

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

T **block()**



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

Key Blocking Operators in the Mono Class

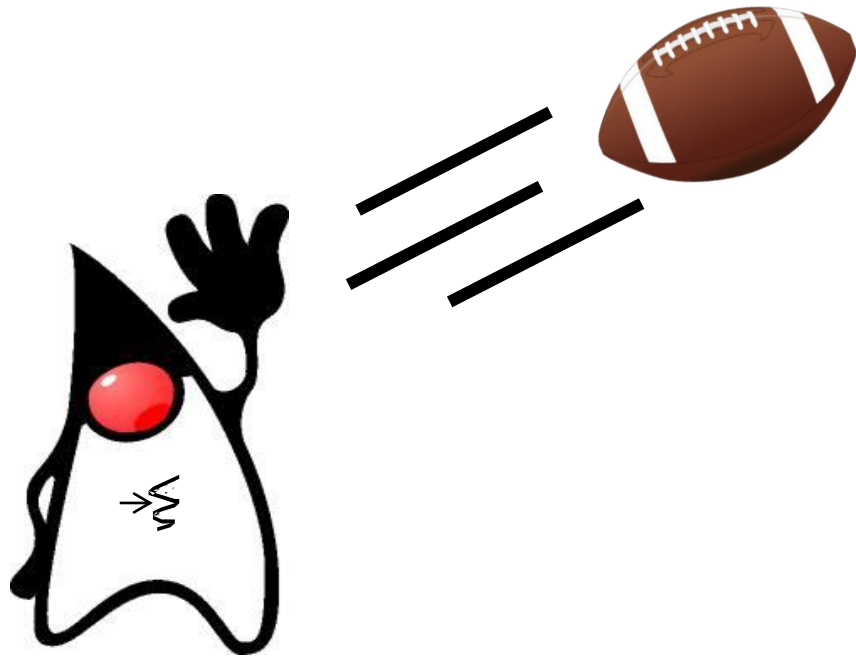
T `block()`

- 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

T `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
 - A checked exception is wrapped in a `RuntimeException`

T `block()`

```
public class RuntimeException  
extends Exception
```

`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

`RuntimeException` and its subclasses are *unchecked exceptions*. Unchecked exceptions do *not* need to be declared in a method or constructor's throws clause if they can be thrown by the execution of the method or constructor and propagate outside the method or constructor boundary.

See 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

T block(**Duration** **timeout**)



See 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

`T block(Duration timeout)`



See 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

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



Key Blocking Operators in the Mono Class

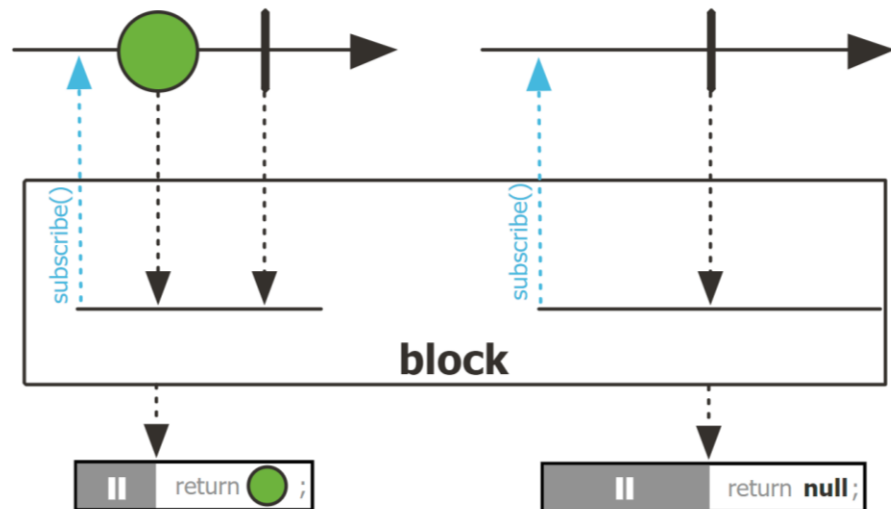
- The `block()` operator
 - Subscribe to this Mono & block until a next signal is received
 - Should only be used if a value is needed before proceeding

```
BigFraction bf1 = ...
```

```
BigFraction bf2 = ...
```

```
BigFraction result = Mono  
    .fromCallable(() -> bf1.multiply(bf2))  
    .subscribeOn(Schedulers.single())  
    .block();
```

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



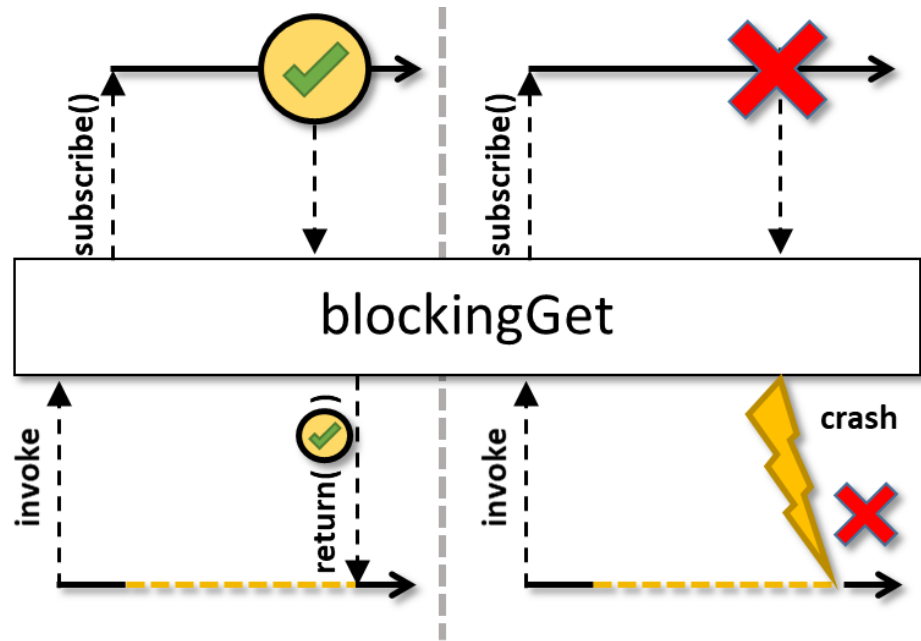
Block caller until the background operation completes

See [Reactive/mono/ex2/src/main/java/MonoEx.java](https://github.com/reactive/reactive-monads/blob/master/src/main/java/MonoEx.java)

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

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



Block caller until the background operation completes

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()`

join

```
public T join()
```

Returns the result value when complete, or throws an (unchecked) exception if completed exceptionally. To better conform with the use of common functional forms, if a computation involved in the completion of this `CompletableFuture` threw an exception, this method throws an (unchecked) `CompletionException` with the underlying exception as its cause.

Returns:

the result value

```
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

`Optional<T> 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

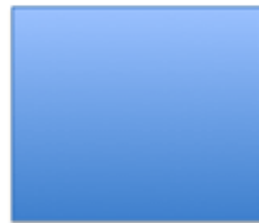
Optional<T> blockOptional()

Optional<Soundcard>



Contains an
object of type
Soundcard

Optional<Soundcard>



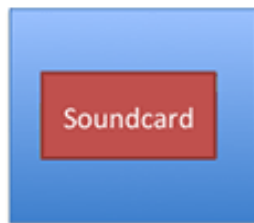
An empty Optional

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
 - Can replace the empty case with an Exception via `Optional.orElseThrow()`

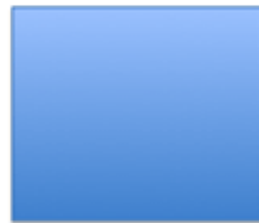
Optional<T> `blockOptional()`

Optional<Soundcard>



Contains an
object of type
Soundcard

Optional<Soundcard>



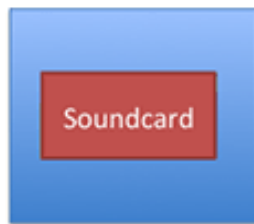
An empty Optional

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
 - Can replace the empty case with an Exception via Optional.orElseThrow()
 - Can return a default value via Optional.orElse() or orElseGet()

Optional<T> blockOptional()

Optional<Soundcard>



Contains an
object of type
Soundcard

Optional<Soundcard>



An empty 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
 - 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

Optional<T> blockOptional()



See 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)`



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 `RuntimeException` is thrown

`Optional<T> blockOptional`
(Duration timeout)

```
public class RuntimeException  
extends Exception
```

`RuntimeException` is the superclass of those exceptions that can be thrown during the normal operation of the Java Virtual Machine.

`RuntimeException` and its subclasses are *unchecked exceptions*. Unchecked exceptions do *not* need to be declared in a method or constructor's throws clause if they can be thrown by the execution of the method or constructor and propagate outside the method or constructor boundary.

See docs.oracle.com/javase/8/docs/api/java/lang/RuntimeException.html

Key Blocking Operators in the Mono Class

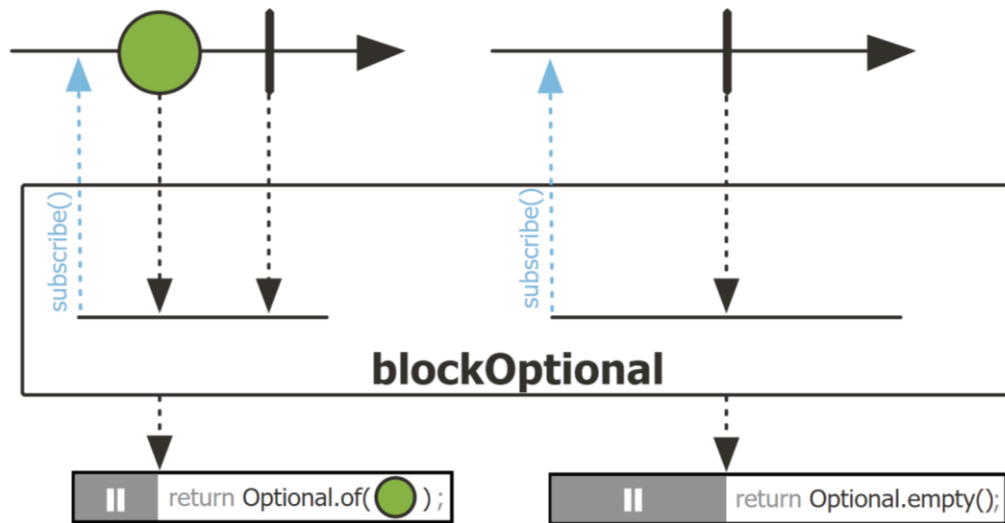
- The `blockOptional()` operator
 - Subscribe to this Mono & block until a next signal is received or the Mono completes empty
 - Should only be used if a value is needed before proceeding

```
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")) ;
```

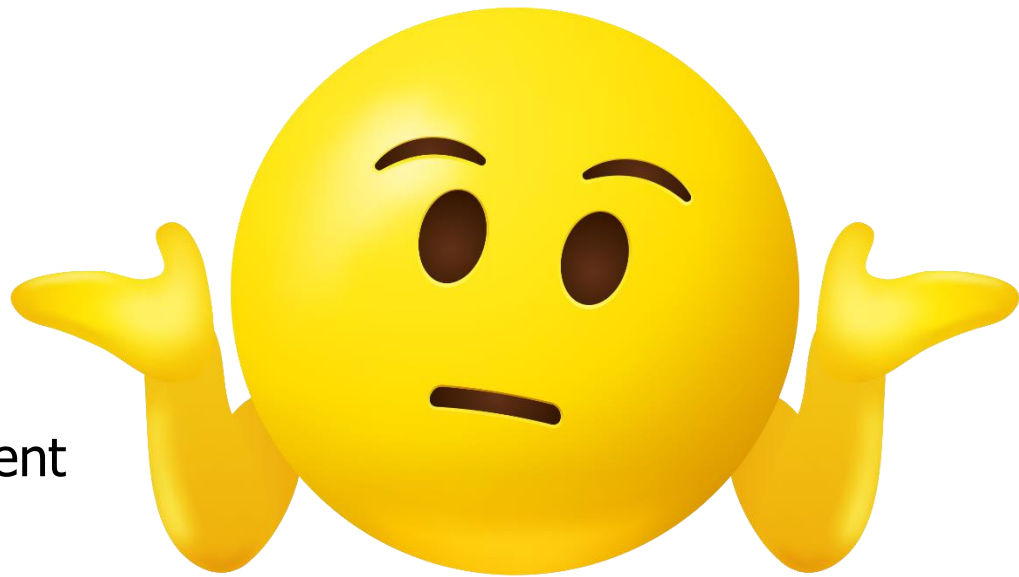


Block caller until the background operation completes

See [Reactive/mono/ex2/src/main/java/MonoEx.java](https://github.com/reactive/reactive-core/blob/master/src/main/java/mono/MonoEx.java)

Key Blocking Operators in the Mono Class

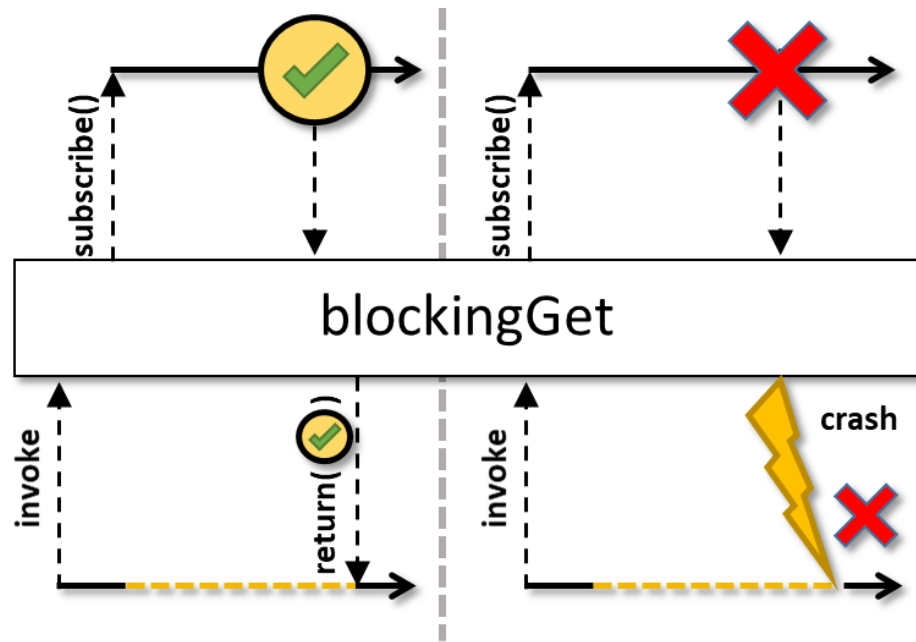
- The `blockOptional()` operator
 - Subscribe to this Mono & block until a next signal is received or the Mono completes empty
 - Should only be used if a value is needed before proceeding
 - There's no direct RxJava equivalent



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

```
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()`

join

```
public T join()
```

Returns the result value when complete, or throws an (unchecked) exception if completed exceptionally. To better conform with the use of common functional forms, if a computation involved in the completion of this `CompletableFuture` threw an exception, this method throws an (unchecked) `CompletionException` with the underlying exception as its cause.

Returns:

the result value

```
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

End of Key Blocking Operators in the Mono Class