Key Action Operators in the Observable Class (Part 2)

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 Observable operators
  - Concurrency & scheduler operators
  - Factory method operators
  - Action operators
    - These operators don't modify an Observable, but instead just use it for side effects
      - e.g., doFinally() & doOnComplete()

Key Action Operators in the Observable Class

Key Action Operators in the Observable Class

- The doFinally() operator
  - Calls the specified Action after the current Observable terminates

```
Observable<T> doFinally
                (Action onFinally)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#doFinally

Key Action Operators in the Observable Class

- The doFinally() operator
  - Calls the specified Action after the current Observable terminates
    - The param is called when Observable signals onError() or onComplete() or is disposed by the downstream

```
Observable<T> doFinally
              (Action onFinally)
```

```
@FunctionalInterface
public interface Action
```

A functional interface similar to Runnable but allows throwing a checked exception.

Method Summary

All Methods	Instance Methods	Abstract Methods

Modifier and Type	Method and Description
void	run() Runs the action and optionally throws a checked exception.

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/functions/Action.html

Key Action Operators in the Observable Class

- The doFinally() operator
  - Calls the specified Action after the current Observable terminates
    - The param is called when Observable signals onError() or onComplete() or is disposed by the downstream
      - Action is a functional interface similar to Runnable but allows throwing a checked exception

```
Observable<T> doFinally
              (Action onFinally)
```

```
@FunctionalInterface
public interface Action
```

A functional interface similar to Runnable but allows throwing a checked exception.

Method Summary

All Methods	Instance Methods	Abstract Methods

Modifier and Type	Method and Description
void	run() Runs the action and optionally throws a checked exception.

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/functions/Action.html

Key Action Operators in the Observable Class

- The doFinally() operator
  - Calls the specified Action after the current Observable terminates
    - The param is called when Observable signals onError() or onComplete() or is disposed by the downstream
      - Action is a functional interface similar to Runnable but allows throwing a checked exception
        - i.e., it is a "callback" that only has side-effects

```
Observable<T> doFinally
              (Action onFinally)
```

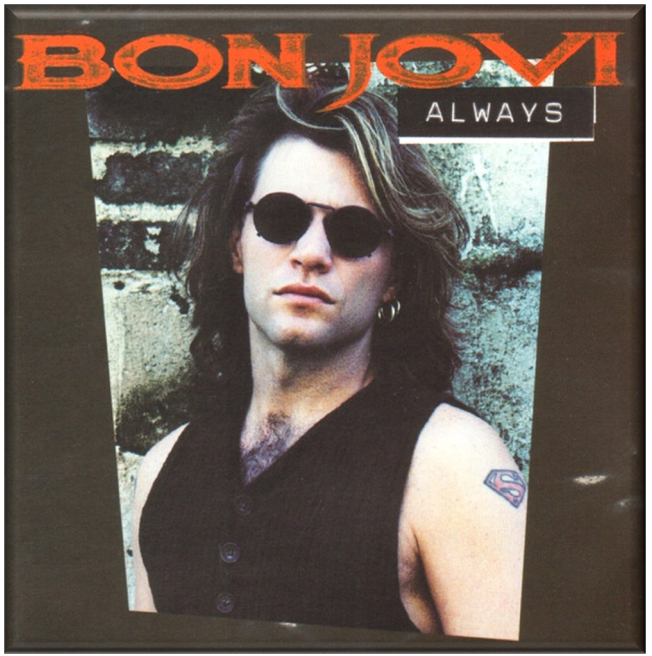
See en.wikipedia.org/wiki/Callback_(computer_programming)

Key Action Operators in the Observable Class

- The doFinally() operator

```
Observable<T> doFinally
              (Action onFinally)
```

  - Calls the specified Action after the current Observable terminates
    - The param is called when Observable signals onError() or onComplete() or is disposed by the downstream
      - Action is a functional interface similar to Runnable but allows throwing a checked exception
    - Action is always called regardless of successful or error completion
      - Similar to a C++ destructor



Contrast this doFinally() behavior with the doOnComplete() behavior

Key Action Operators in the Observable Class

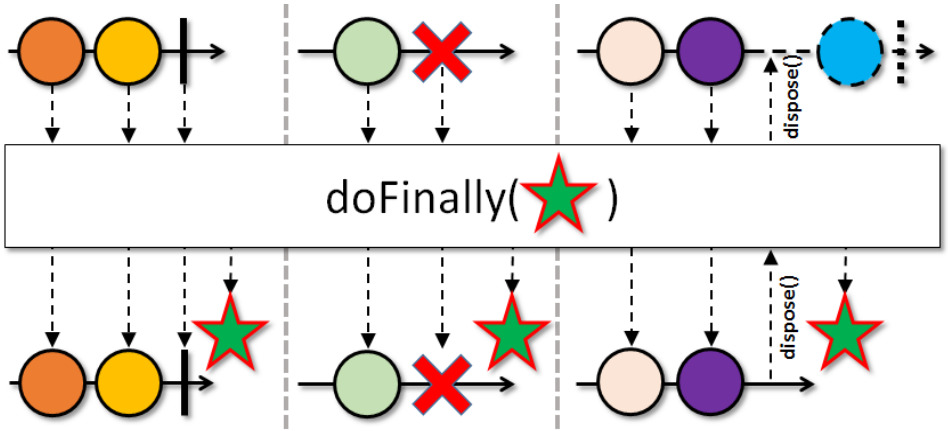
- The doFinally() operator
  - Calls the specified Action after the current Observable terminates
    - The param is called when Observable signals onError() or onComplete() or is disposed by the downstream
    - Returns the new Observable instance

```
Observable<T> doFinally
              (Action onFinally)
```

The type or the value of elements that is processed is unchanged

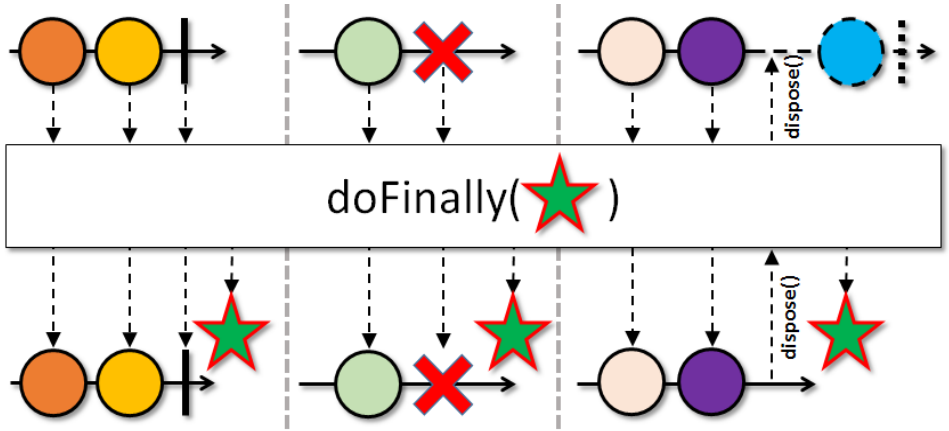
Key Action Operators in the Observable Class

- The doFinally() operator
  - Calls the specified Action after the current Observable terminates
  - Does not operate by default on a particular Scheduler
    - i.e., it uses the current scheduler



Key Action Operators in the Observable Class

- The doFinally() operator
  - Calls the specified Action after the current Observable terminates
  - Does not operate by default on a particular Scheduler
    - i.e., it uses the current scheduler



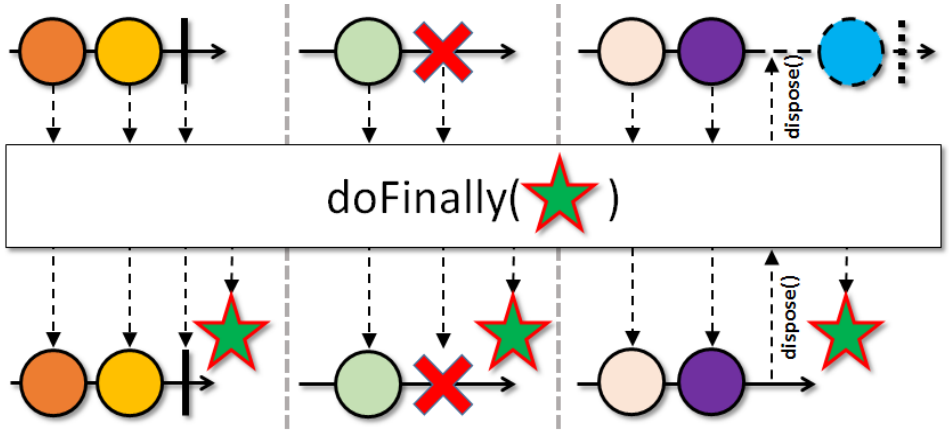

```
Observable
   .create(ObservableEx::emitAsync)
   .observeOn(Schedulers.newThread())
   .map(bigInteger -> ObservableEx
        .checkIfPrime(bigInteger, sb))
   .doFinally(() -> BigFractionUtils.display(sb.toString()))
   ...
```

Print BigInteger objects to aid with debugging

See Reactive/Observable/ex2/src/main/java/ObservableEx.java

Key Action Operators in the Observable Class

- The doFinally() operator
  - Calls the specified Action after the current Observable terminates
  - Does not operate by default on a particular Scheduler
    - i.e., it uses the current scheduler



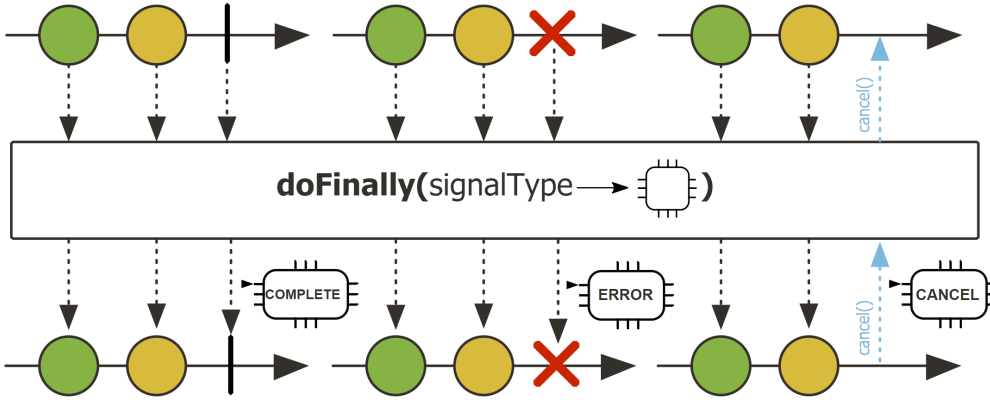

```
Observable
   .create(ObservableEx::emitAsync)
   .observeOn(Schedulers.newThread())
   .map(bigInteger -> ObservableEx
        .checkIfPrime(bigInteger, sb))
   .doFinally(() -> BigFractionUtils.display(sb.toString()))
   ...
```

Only a "side-effect"

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

Key Action Operators in the Observable Class

- The doFinally() operator
  - Calls the specified Action after the current Observable terminates
  - Does not operate by default on a particular Scheduler
  - Project Reactor's operator Flux .doFinally() works the same




```
Scheduler subscriber = Schedulers.newParallel("subscriber", 1);
Flux
   .create(makeAsyncFluxSink())
   .publishOn(subscriber)
   .doFinally(__ -> subscriber
              .dispose()) ...
```

Only a "side-effect"

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#doFinally

Key Action Operators in the Observable Class

- The doFinally() operator
  - Calls the specified Action after the current Observable terminates
  - Does not operate by default on a particular Scheduler
  - Project Reactor's operator Flux .doFinally() works the same
- The Java Streams framework has no operations like doFinally()
  - Any cleanup can be done after the stream's terminal operation completes synchronously

Interface Stream<T>

Type Parameters:

`T` - the type of the stream elements

All Superinterfaces:

`AutoCloseable`, `BaseStream<T,Stream<T>>`

```
public interface Stream<T>
extends BaseStream<T,Stream<T>>
```

A sequence of elements supporting sequential and parallel aggregate operations. The following example illustrates an aggregate operation using `Stream` and `IntStream`:

```
int sum = widgets.stream()
                 .filter(w -> w.getColor() == RED)
                 .mapToInt(w -> w.getWeight())
                 .sum();
```

In this example, `widgets` is a `Collection<Widget>`. We create a stream of `Widget` objects via `Collection.stream()`, filter it to produce a stream containing only the red widgets, and then transform it into a stream of `int` values representing the weight of each red widget. Then this stream is summed to produce a total weight.

In addition to `Stream`, which is a stream of object references, there are primitive specializations for `IntStream`, `LongStream`, and `DoubleStream`, all of which are referred to as "streams" and conform to the characteristics and restrictions described here.

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html

Key Action Operators in the Observable Class

- The doOnComplete() operator
  - Calls the specified Action after the current Observable completes

```
Observable<T> doOnComplete
                (Action onComplete)
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html#doOnComplete

Key Action Operators in the Observable Class

- The doOnComplete() operator
  - Calls the specified Action after the current Observable completes
    - The Action parameter is called when the Observable signals onComplete()
      - Action is a functional interface similar to Runnable but allows throwing a checked exception

```
Observable<T> doOnComplete
              (Action onComplete)
```

```
@FunctionalInterface
public interface Action
```

A functional interface similar to Runnable but allows throwing a checked exception.

Method Summary

All Methods	Instance Methods	Abstract Methods

Modifier and Type	Method and Description
void	run() Runs the action and optionally throws a checked exception.

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/functions/Action.html

Key Action Operators in the Observable Class

- The doOnComplete() operator
  - Calls the specified Action after the current Observable completes
    - The Action parameter is called when the Observable signals onComplete()
      - Action is a functional interface similar to Runnable but allows throwing a checked exception
        - i.e., again, it's a callback that only has side-effects

```
Observable<T> doOnComplete
              (Action onComplete)
```

See en.wikipedia.org/wiki/Callback_(computer_programming)

Key Action Operators in the Observable Class

- The doOnComplete() operator
  - Calls the specified Action after the current Observable completes
    - The Action parameter is called when the Observable signals onComplete()
      - Action is a functional interface similar to Runnable but allows throwing a checked exception
      - Action is called only on successful completion, but not when errors occur

```
Observable<T> doOnComplete
              (Action onComplete)
```

Contrast this doOnComplete() behavior with the doFinally() behavior

Key Action Operators in the Observable Class

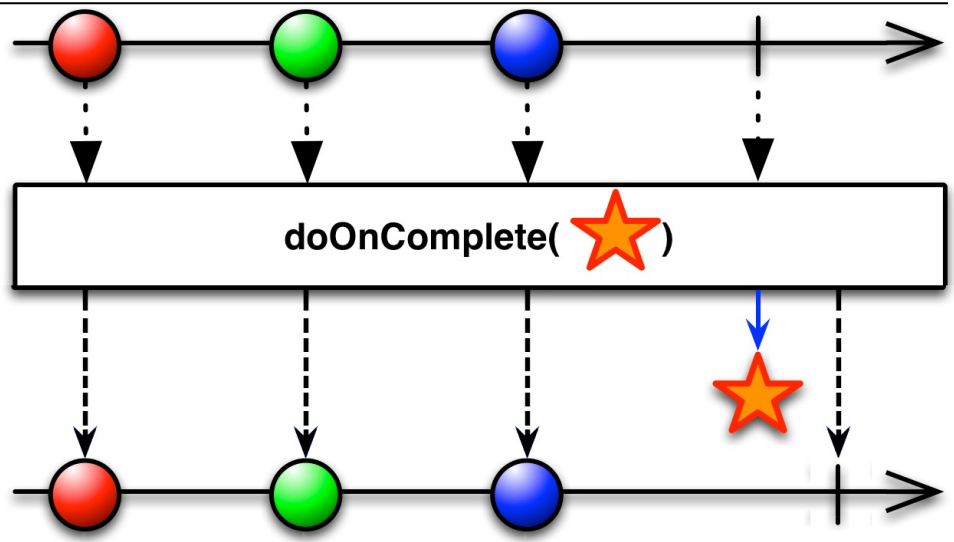
- The doOnComplete() operator
  - Calls the specified Action after the current Observable completes
    - The Action parameter is called when the Observable signals onComplete()
    - Returns the new Observable instance

```
Observable<T> doOnComplete
              (Action onComplete)
```

Can't change the type or the value of elements it processes

Key Action Operators in the Observable Class

- The doOnComplete() operator
  - Calls the specified Action after the current Observable completes
  - doOnComplete() does not operate by default on a particular Scheduler



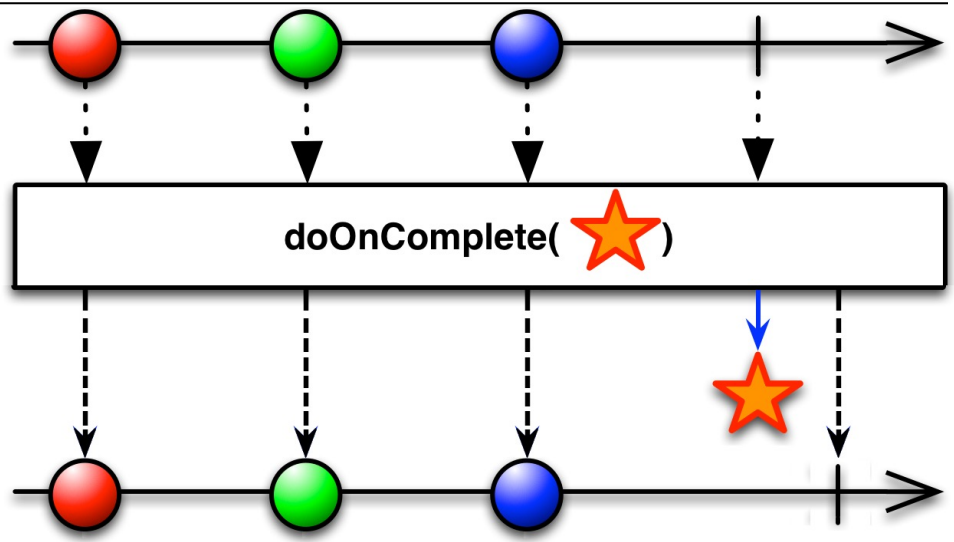

```
Observable
   .create(ObservableEx::emitInterval)
   .map(bigInt -> ObservableEx.checkIfPrime(bigInt, sb))
   .doOnComplete(() -> BigFractionUtils.display(sb.toString()))
   ...
```

Print BigIntegers to aid debugging

See Reactive/Observable/ex2/src/main/java/ObservableEx.java

Key Action Operators in the Observable Class

- The doOnComplete() operator
  - Calls the specified Action after the current Observable completes
  - doOnComplete() does not operate by default on a particular Scheduler



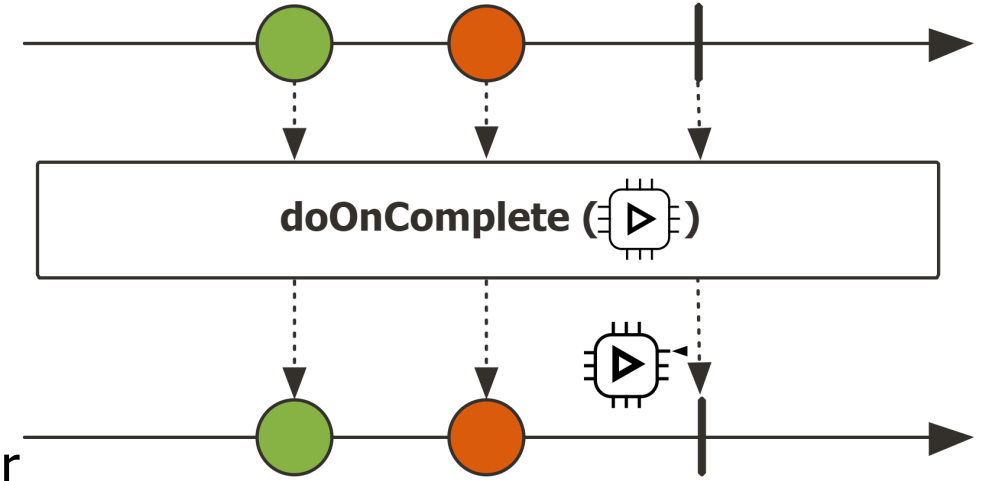

```
Observable
   .create(ObservableEx::emitInterval)
   .map(bigInt -> ObservableEx.checkIfPrime(bigInt, sb))
   .doOnComplete(() -> BigFractionUtils.display(sb.toString()))
   ...
```

Only a "side-effect"

See Reactive/Observable/ex2/src/main/java/ObservableEx.java

Key Action Operators in the Observable Class

- The doOnComplete() operator
  - Calls the specified Action after the current Observable completes
  - doOnComplete() does not operate by default on a particular Scheduler
  - The Flux.doOnComplete() operator in Project Reactor works the same




```
Flux
   .create(makeAsyncFluxSink())
   ...
   .map(bigInt -> FluxEx.checkIfPrime(bigInt, sb))
   .doOnComplete(() -> BigFractionUtils.display(sb.toString()))
   ...
```

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#doOnComplete

End of Key Action Operators in the Observable Class (Part 2)