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

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

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

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

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.

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

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

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
      - Action is always called regardless of successful or error completion
        - Similar to a C++ destructor

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

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

\[
\text{Observable}\langle T \rangle \ \text{doFinally} \ (\text{Action onFinally})
\]

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


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
    - 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

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

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
doFinally)
  .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

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html](http://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html)
Key Action Operators in the Observable Class

- The doOnComplete() operator
  
  ```java
  Observable<T> doOnComplete(
      Action onComplete)
  ```

- Calls the specified Action after the current Observable completes

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
)
```

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

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

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

```java
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

```java
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](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

```java
Flutter
    .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](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#doOnComplete)
End of Key Action Operators in the Observable Class (Part 2)