Key Suppressing Operators in the Flux 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 Flux operators
  - Concurrency operators
  - Scheduler operators
  - Factory method operators
  - Action operators
  - Suppressing operators
    - These operators create a Flux and/or Mono that changes or ignores (portions of) its payload
      - e.g., filter(), take(), & then()

Key Suppressing Operators in the Flux Class

Key Suppressing Operators in the Flux Class

- The filter() operator
  - Evaluate each source value against the given Predicate

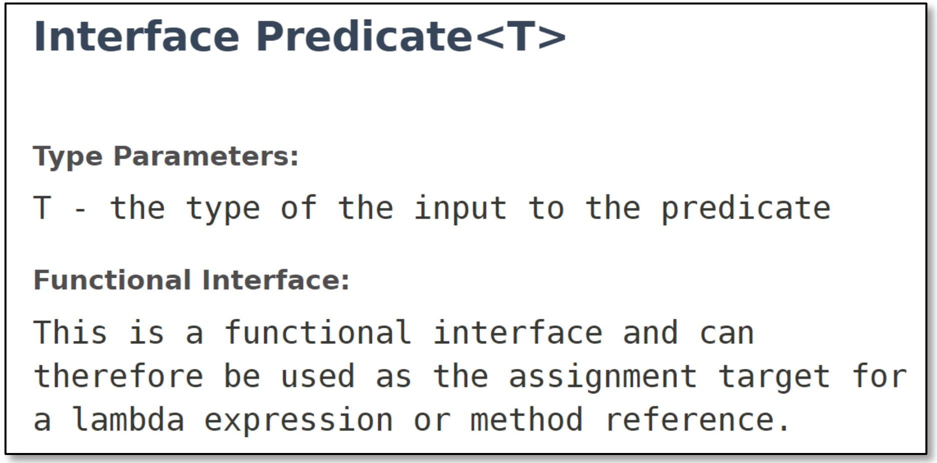
```
Flux<T> filter
   (Predicate<? super T> p)
```

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

Key Suppressing Operators in the Flux Class

- The filter() operator
  - Evaluate each source value against the given Predicate
    - If the predicate test succeeds, the value is emitted

```
Flux<T> filter
   (Predicate<? super T> p)
```



Interface Predicate<T>

Type Parameters:

T - the type of the input to the predicate

Functional Interface:

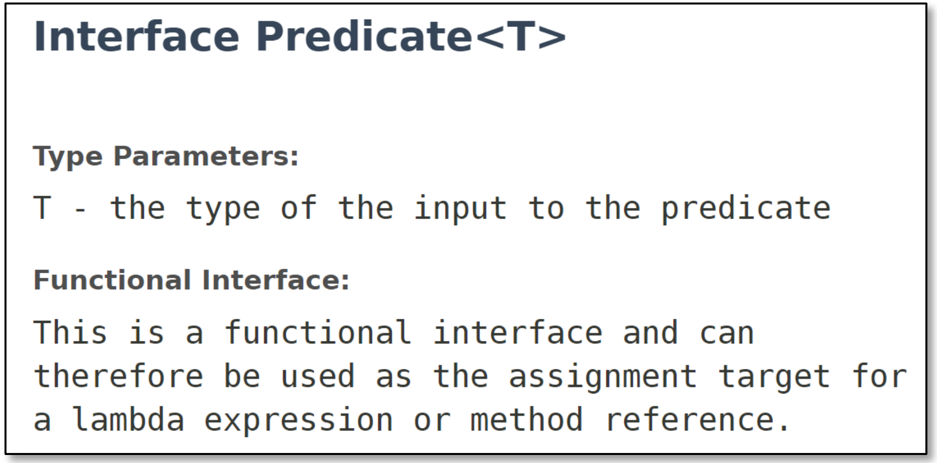
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

See docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html

Key Suppressing Operators in the Flux Class

- The filter() operator
  - Evaluate each source value against the given Predicate
    - If the predicate test succeeds, the value is emitted
    - If the predicate test fails, the value is ignored & a request of 1 is made upstream

```
Flux<T> filter
   (Predicate<? super T> p)
```



Interface Predicate<T>

Type Parameters:

`T - the type of the input to the predicate`

Functional Interface:

`This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.`

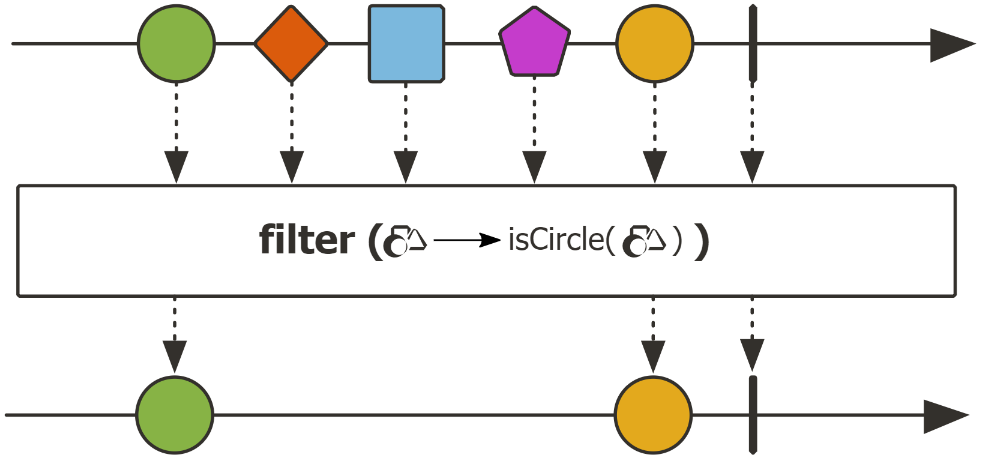
Key Suppressing Operators in the Flux Class

- The filter() operator
  - Evaluate each source value against the given Predicate
    - If the predicate test succeeds, the value is emitted
    - If the predicate test fails, the value is ignored & a request of 1 is made upstream
    - Returns a new Flux containing only the values that pass the predicate test

```
Flux<T> filter
  (Predicate<? super T> p)
```

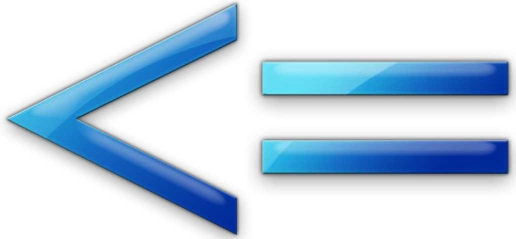
Key Suppressing Operators in the Flux Class

- The filter() operator
  - Evaluate each source value against the given Predicate
  - The # of output elements may be less than # of input elements



```
Flux
   .range(1, sMAX_ITERATIONS)
   ...
   .map(sGenerateRandomBigInteger)
   .filter(bigInteger -> !bigInteger
                .mod(BigInteger.TWO)
                .equals(BigInteger.ZERO))

   .subscribe(...);
```

Only emit odd numbers

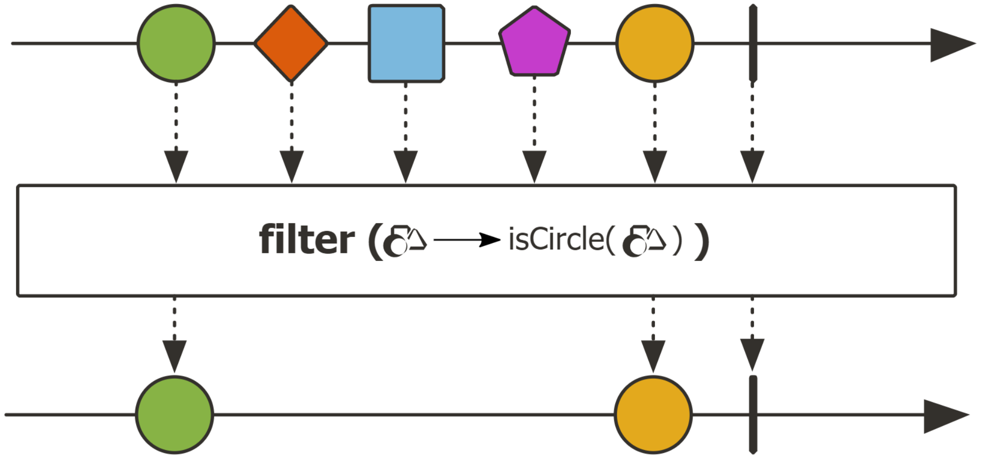


See [Reactive/flux/ex2/src/main/java/FluxEx.java]

Key Suppressing Operators in the Flux Class

- The filter() operator
  - Evaluate each source value against the given Predicate
  - The # of output elements may be less than # of input elements

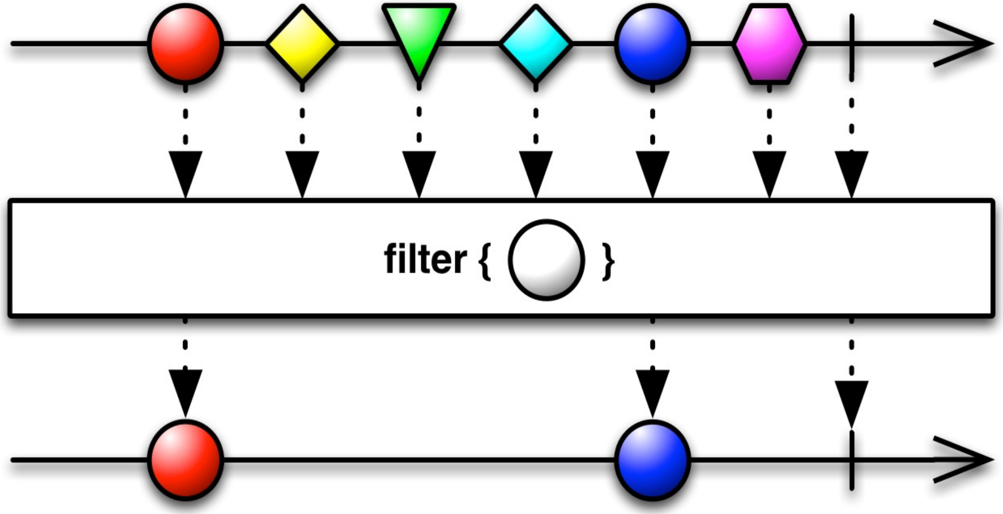
```
Flux
   .range(1, sMAX_ITERATIONS)
   ...
   .map(sGenerateRandomBigInteger)
   .filter(bigInteger -> !bigInteger
                .mod(BigInteger.TWO)
                .equals(BigInteger.ZERO))

   .subscribe(...);
```



filter() can't change the type or value of elements it processes

Key Suppressing Operators in the Flux Class

- The filter() operator
  - Evaluate each source value against the given Predicate
  - The # of output elements may be less than # of input elements
  - RxJava's Observable.filter() works the same way



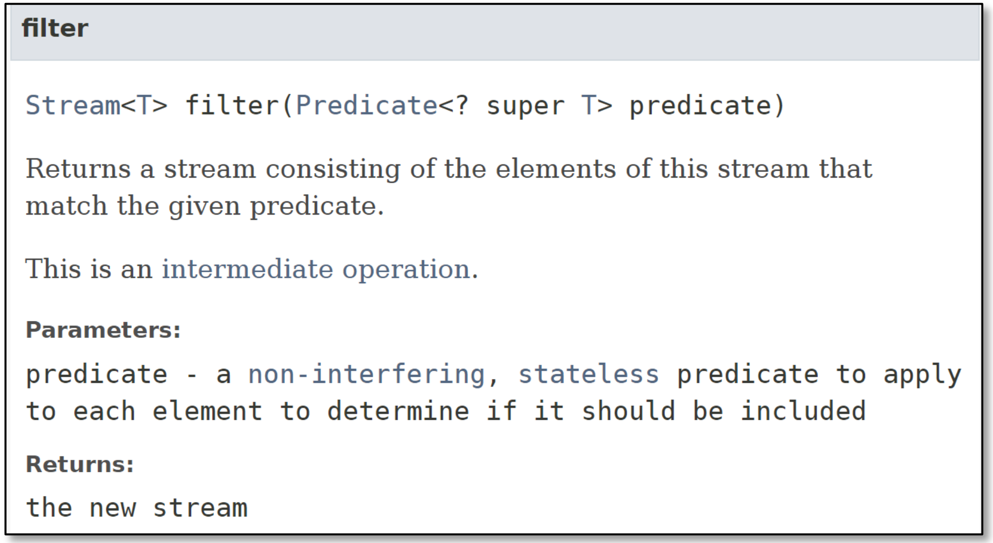
```
Observable
   .range(1, sMAX_ITERATIONS)
   ...
   .map(sGenerateRandomBigInteger)
   .filter(bigInteger -> !bigInteger.mod(BigInteger.TWO)
                                  .equals(BigInteger.ZERO))
   .subscribe(...);
```

Only emit odd #'s

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

Key Suppressing Operators in the Flux Class

- The filter() operator
  - Evaluate each source value against the given Predicate
  - The # of output elements may be less than # of input elements
  - RxJava's Observable.filter() works the same way
  - Similar to Java Streams filter() operation



filter

`Stream<T> filter(Predicate<? super T> predicate)`

Returns a stream consisting of the elements of this stream that match the given predicate.

This is an intermediate operation.

Parameters:

`predicate - a non-interfering, stateless predicate to apply to each element to determine if it should be included`

Returns:

`the new stream`

```
List<Double> oddNumbers =
  LongStream
    .rangeClosed(1, 100)
    .filter(n -> (n & 1) != 0)
    .toList();
```

Only emit odd #'s

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

Key Suppressing Operators in the Flux Class

- The take() operator
  - Take only the first N values from this Flux, if available

```
Flux<T> take(long n)
```

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

Key Suppressing Operators in the Flux Class

- The take() operator
  - Take only the first N values from this Flux, if available
    - The param is the # of items to emit from this Flux

```
Flux<T> take(long n)
```

Key Suppressing Operators in the Flux Class

- The take() operator
  - Take only the first N values from this Flux, if available
    - The param is the # of items to emit from this Flux
    - Returns a Flux limited to size 'n'

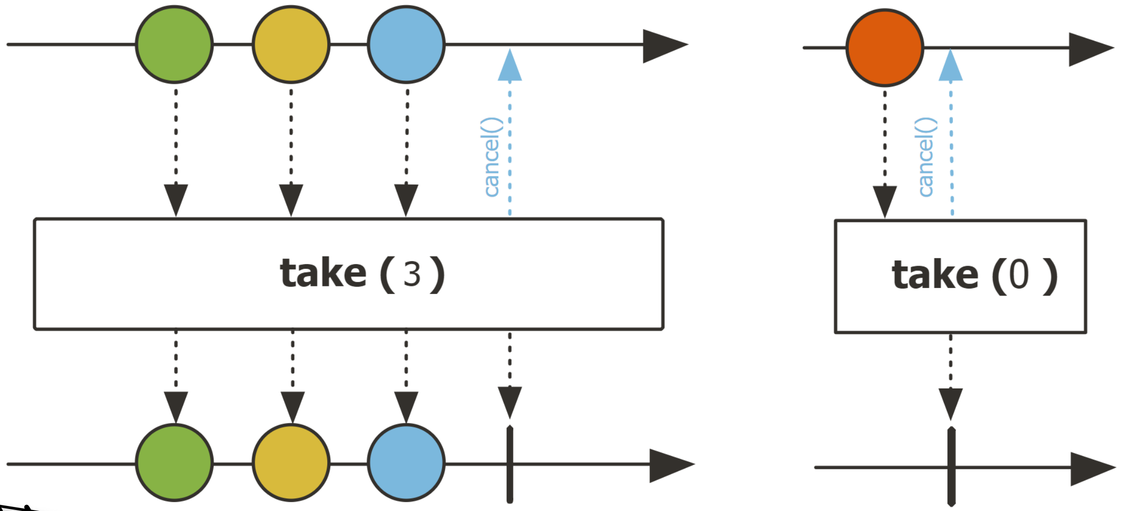
```
Flux<T> take(long n)
```

See thedailyguardian.com/the-procrustean-bed

Key Suppressing Operators in the Flux Class

- The take() operator
  - Take only the first N values from this Flux, if available
  - Used to limit otherwise "infinite" streams

```
Flux
    .interval
        (sSLEEP.toMillis())
    ...
    .take(sMAX_ITERATIONS)
    ...
```



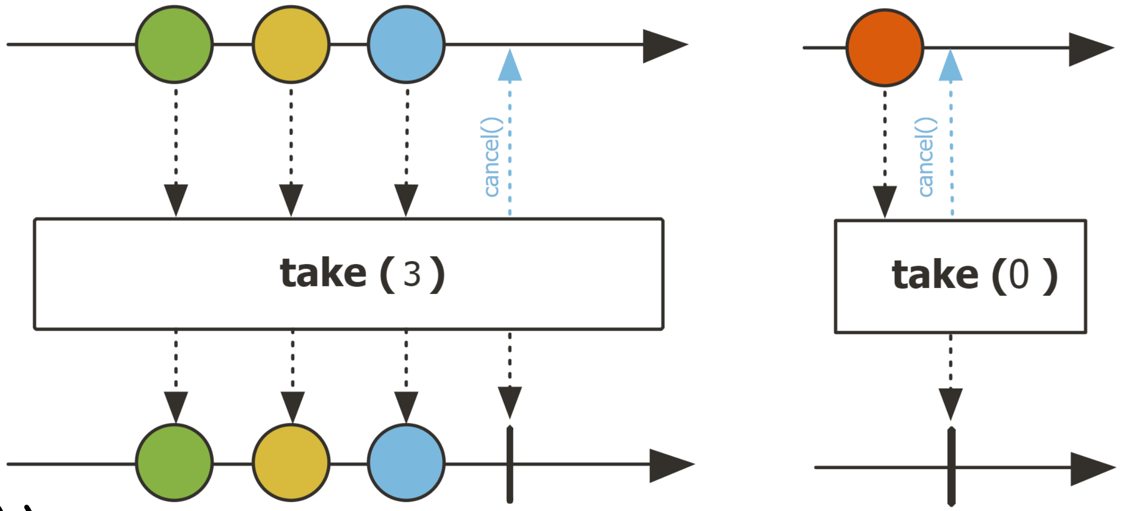
Generate an infinite series of integers periodically in a background thread

See earlier discussion of the Flux.interval() operator

Key Suppressing Operators in the Flux Class

- The take() operator
  - Take only the first N values from this Flux, if available
  - Used to limit otherwise "infinite" streams

```
Flux
   .interval
      (sSLEEP.toMillis())
   ...
   .take(sMAX_ITERATIONS)
   ...
```

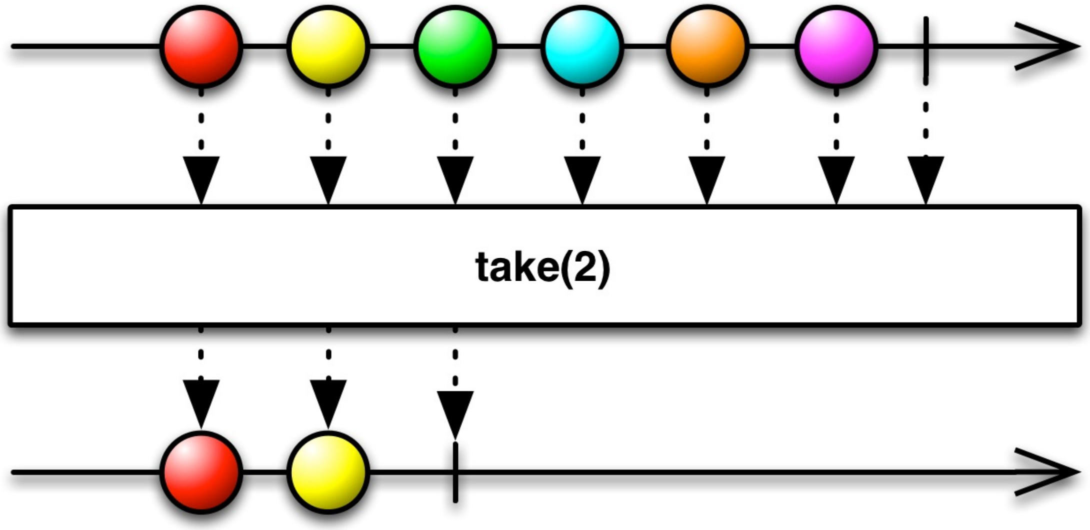


Only process sMAX_ITERATIONS # of emitted values from interval()

See Reactive/Flux/ex2/src/main/java/FluxEx.java

Key Suppressing Operators in the Flux Class

- The take() operator
  - Take only the first N values from this Flux, if available
  - Used to limit otherwise "infinite" streams
  - RxJava's Observable.take() works the same



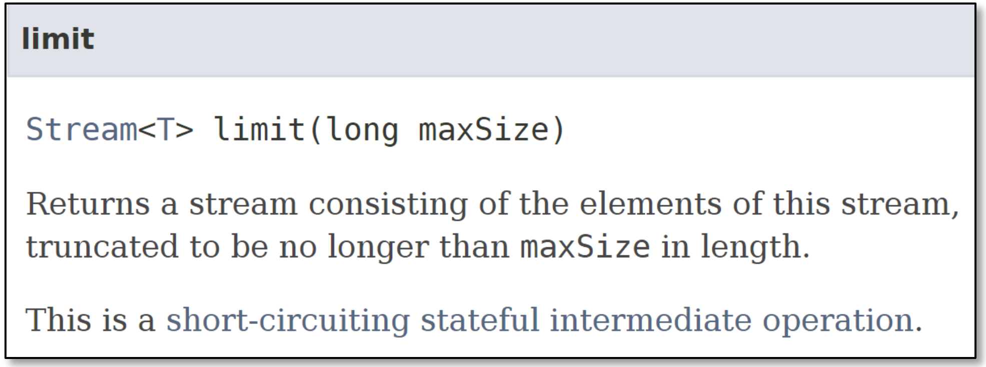
```
Observable
  .interval(sSLEEP_DURATION,
            MILLISECONDS)
  ...
  .take(sMAX_ITERATIONS)
```

Stop emitting after sMAX_ITERATIONS

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

Key Suppressing Operators in the Flux Class

- The take() operator
  - Take only the first N values from this Flux, if available
  - Used to limit otherwise "infinite" streams
  - RxJava's Observable.take() works the same
  - Similar to Stream.limit() in Java Streams



limit

`Stream<T> limit(long maxSize)`

Returns a stream consisting of the elements of this stream, truncated to be no longer than `maxSize` in length.

This is a short-circuiting stateful intermediate operation.

```
List<Double> oddNumbers = Stream
   .iterate(1L, l -> l + 1)
   .filter(n -> (n & 1) != 0)
   .limit(100)
   .toList();
```

Only emit 100 odd #'s

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

Key Suppressing Operators in the Flux Class

- The then() operator
  - Let this Flux complete & then play signals from a provided Mono

```
<V> Mono<V> then(Mono<V> other)
```

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

Key Suppressing Operators in the Flux Class

```
<V> Mono<V> then(Mono<V> other)
```

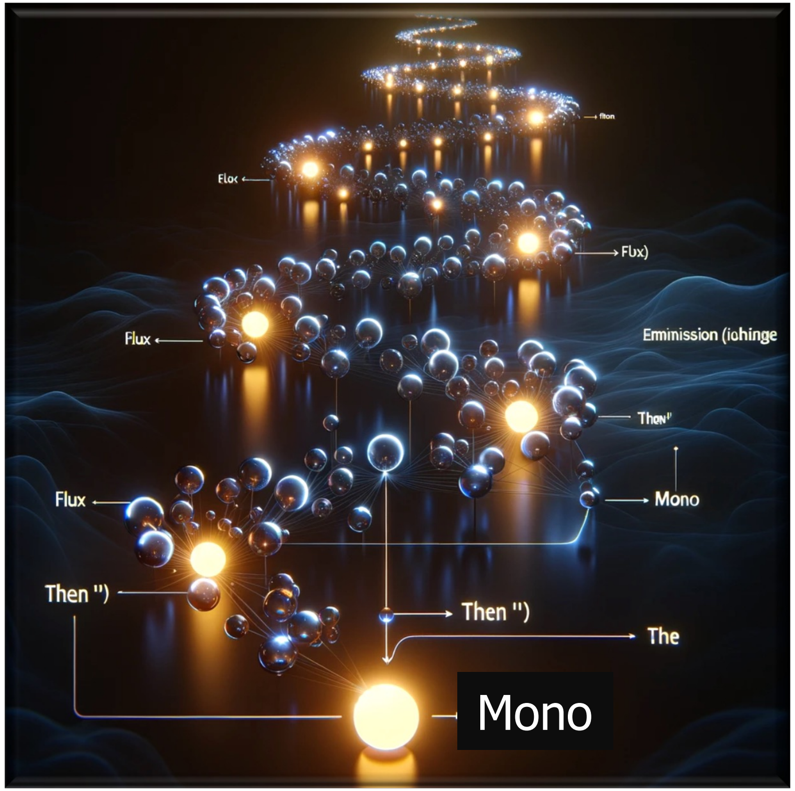
- The then() operator
  - Let this Flux complete & then play signals from a provided Mono
    - The param a Mono to emit from after termination
      - i.e., ignore elements from this Flux & transform its completion signal into the emission & completion signal of a provided Mono<V>

Key Suppressing Operators in the Flux Class

- The then() operator

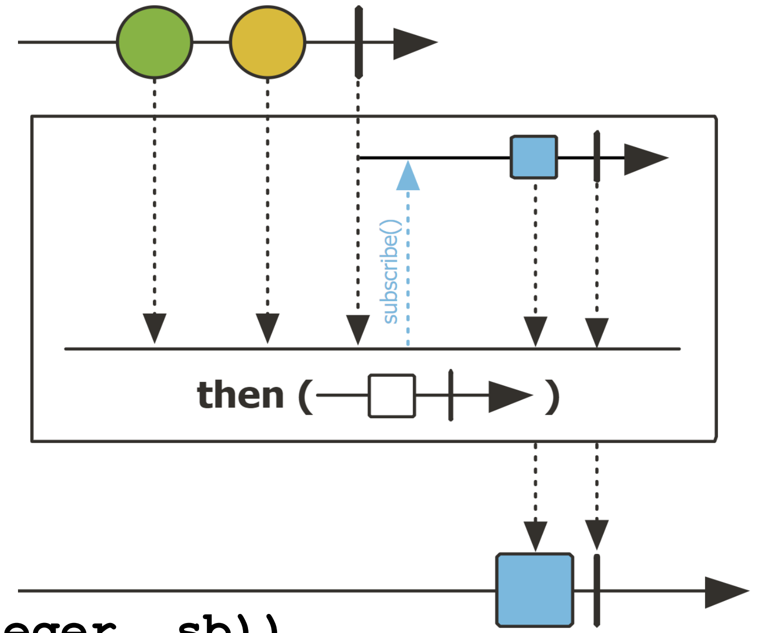
`<V> Mono<V> then(Mono<V> other)`

  - Let this Flux complete & then play signals from a provided Mono
    - The param a Mono to emit from after termination
    - Returns a new Flux that waits for source completion & then emits from the supplied Mono



Key Suppressing Operators in the Flux Class

- The then() operator
  - Let this Flux complete & then play signals from a provided Mono
  - This "data-suppressing" operator replaces its payload with another



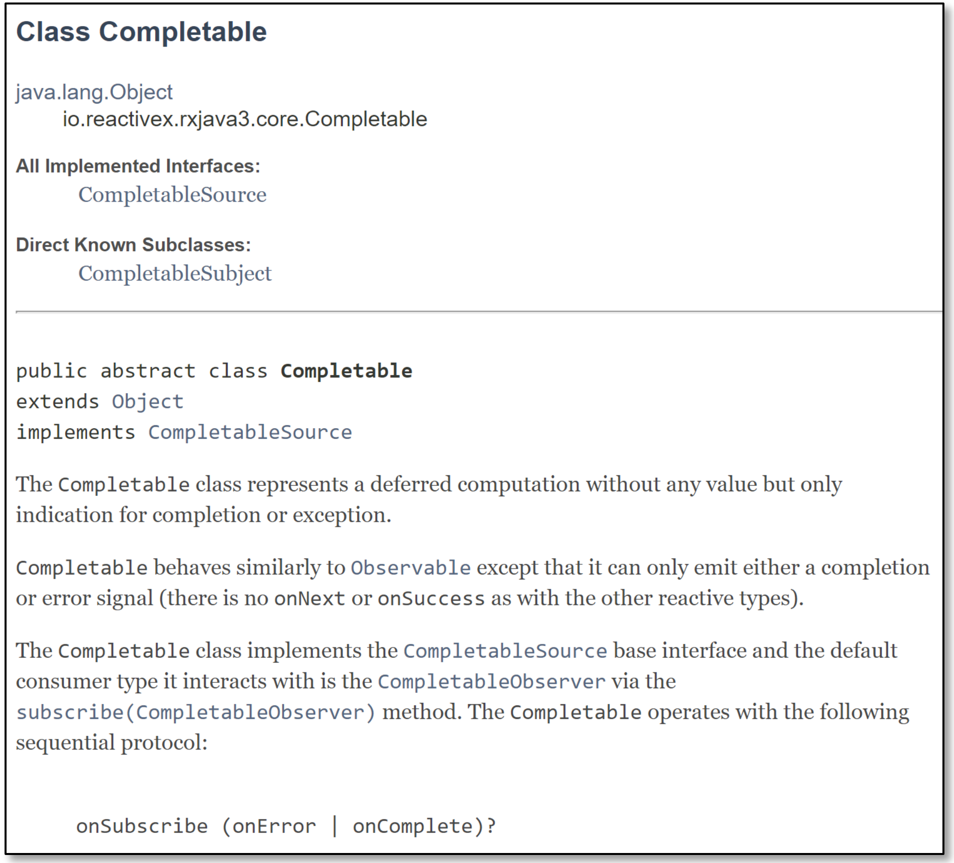
```
return Flux
    .create(makeTimedFluxSink())
    .doOnNext(...)
    .map(bigInteger ->
         FluxEx.checkIfPrime(bigInteger, sb))
    .doOnNext(...)
    .then(Mono
           .fromRunnable(() ->
                         BigFractionUtils.display(sb.toString())));
```

Display results & indicate an async operation completed

See Reactive/flux/ex2/src/main/java/FluxEx.java

Key Suppressing Operators in the Flux Class

- The then() operator
  - Let this Flux complete & then play signals from a provided Mono
  - This "data-suppressing" operator replaces its payload with another
  - RxJava doesn't really have an equivalent, though Completable can be used in a similar way



Class Completable

java.lang.Object
    io.reactivex.rxjava3.core.Completable

All Implemented Interfaces:
    CompletableSource

Direct Known Subclasses:
    CompletableSubject

```
public abstract class Completable
extends Object
implements CompletableSource
```

The `Completable` class represents a deferred computation without any value but only indication for completion or exception.

`Completable` behaves similarly to `Observable` except that it can only emit either a completion or error signal (there is no `onNext` or `onSuccess` as with the other reactive types).

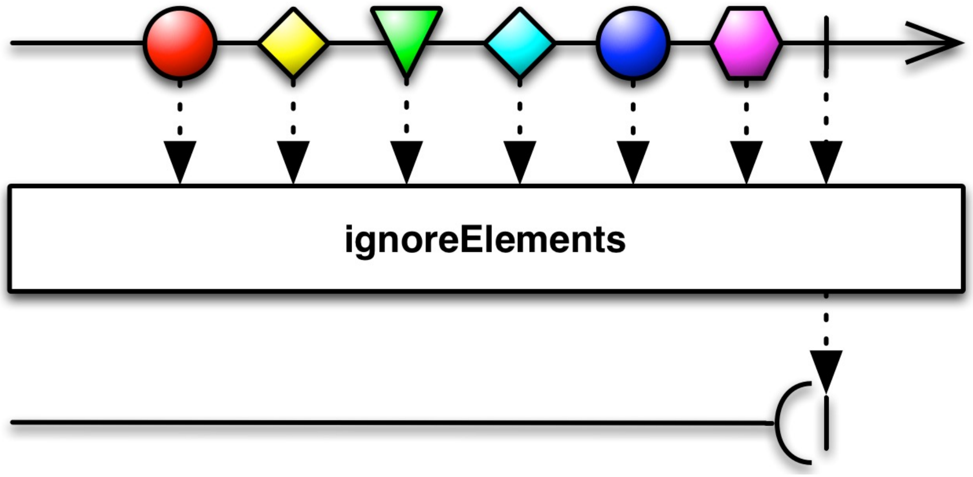
The `Completable` class implements the `CompletableSource` base interface and the default consumer type it interacts with is the `CompletableObserver` via the `subscribe(CompletableObserver)` method. The `Completable` operates with the following sequential protocol:

```
    onSubscribe (onError | onComplete)?
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Completable.html

Key Suppressing Operators in the Flux Class

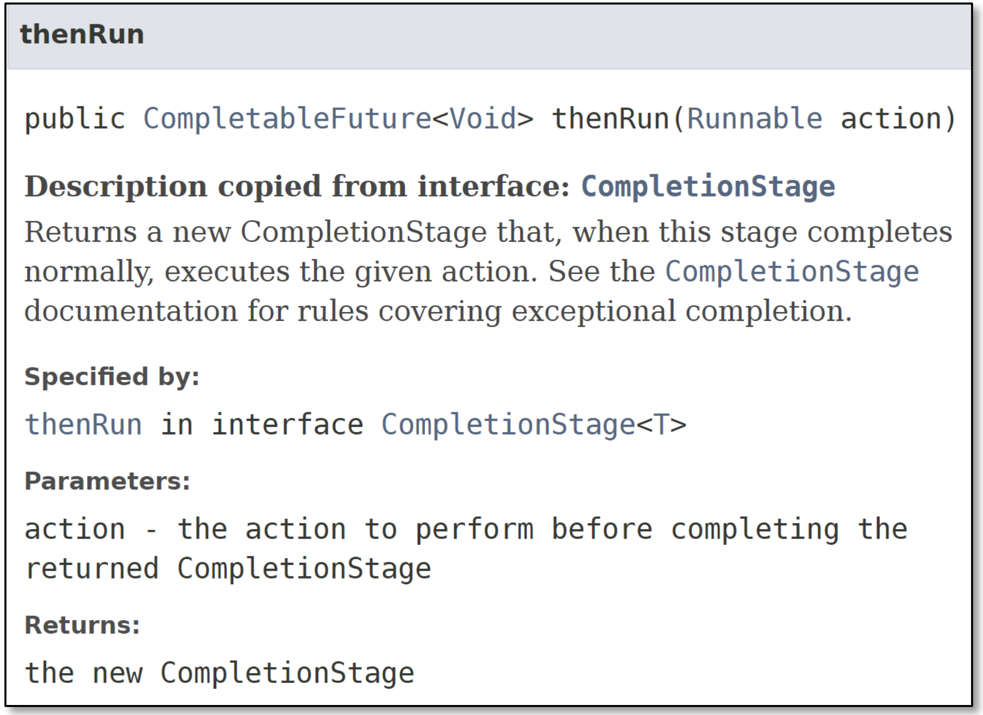
- The then() operator
  - Let this Flux complete & then play signals from a provided Mono
  - This "data-suppressing" operator replaces its payload with another
  - RxJava doesn't really have an equivalent, though Completable can be used in a similar way
    - Created via Observable.ignoreElements()
      - Returns a Completable that ignores the success value of this Observable & signals onComplete() or onError()



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

Key Suppressing Operators in the Flux Class

- The then() operator
  - Let this Flux complete & then play signals from a provided Mono
  - This "data-suppressing" operator replaces its payload with another
  - RxJava doesn't really have an equivalent, though Completable can be used in a similar way
  - Similar to Java Completable Future thenRun()



thenRun

```
public CompletableFuture<Void> thenRun(Runnable action)
```

Description copied from interface: `CompletionStage`

Returns a new CompletionStage that, when this stage completes normally, executes the given action. See the `CompletionStage` documentation for rules covering exceptional completion.

Specified by:

`thenRun in interface CompletionStage<T>`

Parameters:

`action - the action to perform before completing the returned CompletionStage`

Returns:

`the new CompletionStage`

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#thenRun

End of Key Suppressing Operators in the Flux Class