Key Suppressing Operators in the Observable 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 Observable operators
  - Concurrency & scheduler operators
  - Factory method operators
  - Action operators
  - Suppressing operators
    - These operators create an Observable and/or Single that changes or ignores (portions of) its payload
      - e.g., filter(), take(), & ignoreElements()

Key Suppressing Operators in the Observable Class

Key Suppressing Operators in the Observable Class

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

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

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

Key Suppressing Operators in the Observable Class

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

```
Observable<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 reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/functions/Predicate.html

Key Suppressing Operators in the Observable Class

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

```
Observable<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.

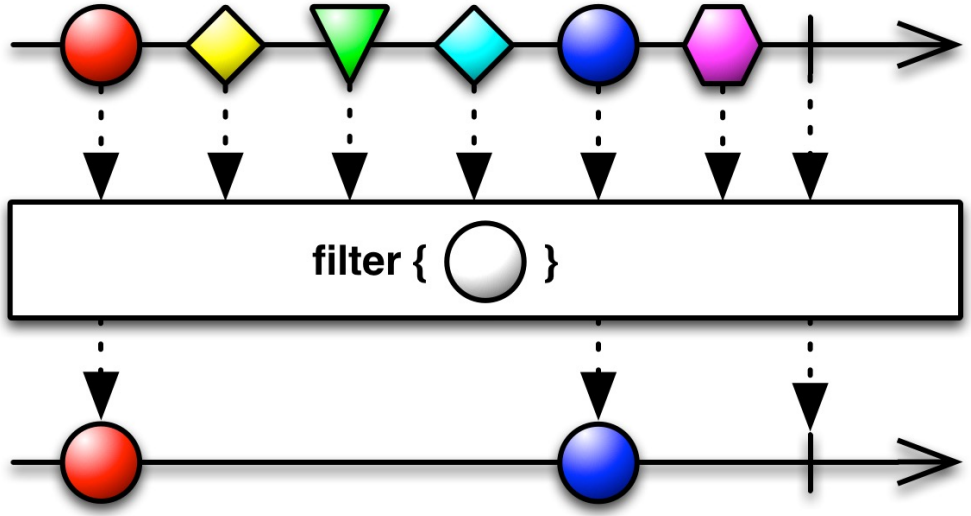
Key Suppressing Operators in the Observable Class

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

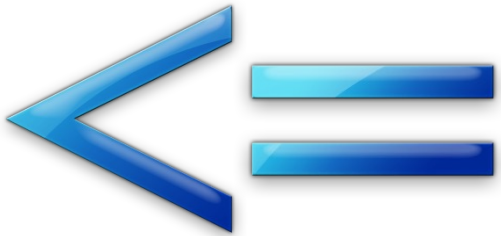
```
Observable<T> filter
   (Predicate<? super T> p)
```

Key Suppressing Operators in the Observable Class

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



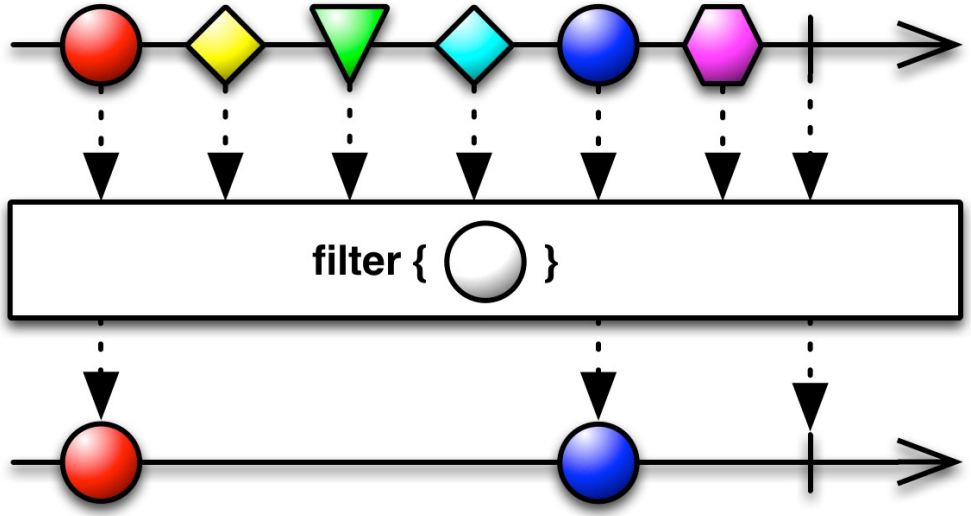

```
Observable
   .rangeLong(1, sMAX_ITERS)
   ...
   .map(sGenerateRandomBigInteger)
   .filter(bigInteger -> !bigInteger
            .mod(BigInteger.TWO)
            .equals(BigInteger.ZERO))


   .subscribe(...);
```



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

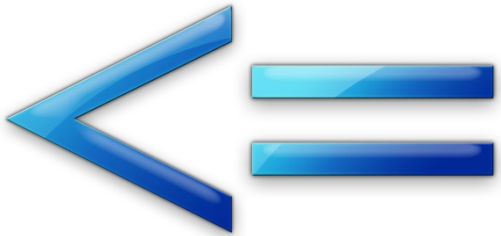
Key Suppressing Operators in the Observable Class

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




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

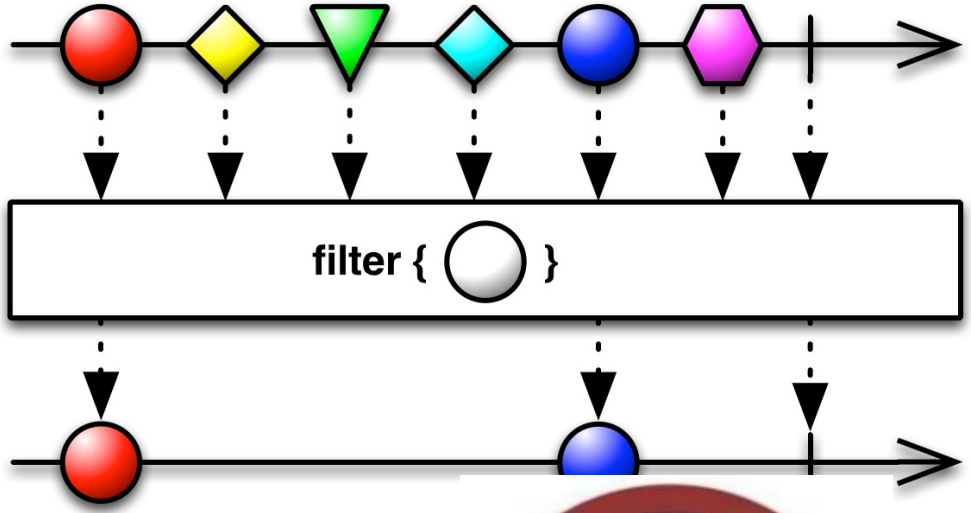
Only emit odd numbers



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

Key Suppressing Operators in the Observable Class

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

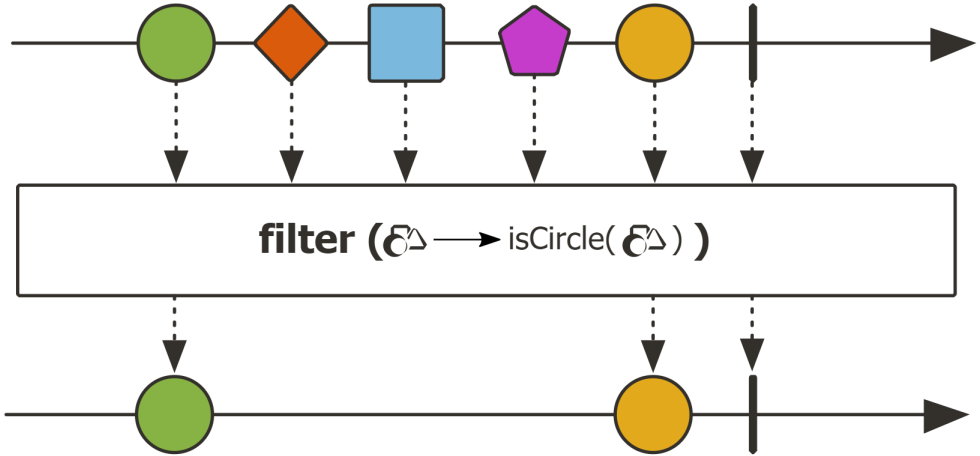



```
Observable
   .rangeLong(1, sMAX_ITERS)
   ...
   .map(sGenerateRandomBigInteger)
   .filter(bigInteger -> !bigInteger
            .mod(BigInteger.TWO)
            .equals(BigInteger.ZERO))


   .subscribe(...);
```

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

Key Suppressing Operators in the Observable Class

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




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

Key Suppressing Operators in the Observable Class

- The filter() operator
  - Evaluate each source value against the given Predicate
  - The # of output elements may be < than # of input elements
  - Project Reactor's Flux.filter() operator works the same way
  - Similar to Stream.filter() method in Java Streams

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`

Only emit odd #'s

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

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

Key Suppressing Operators in the Observable Class

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

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

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

Key Suppressing Operators in the Observable Class

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

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

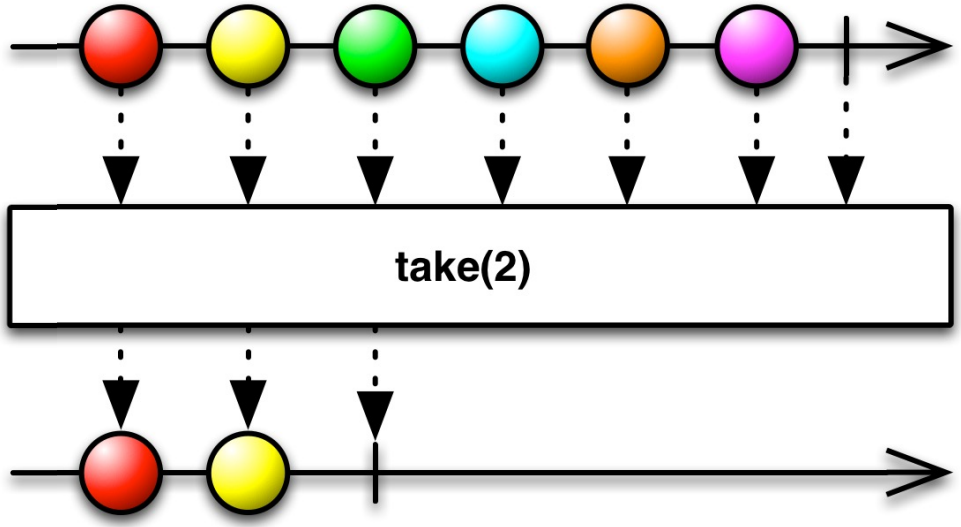
Key Suppressing Operators in the Observable Class

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

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

Key Suppressing Operators in the Observable Class

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



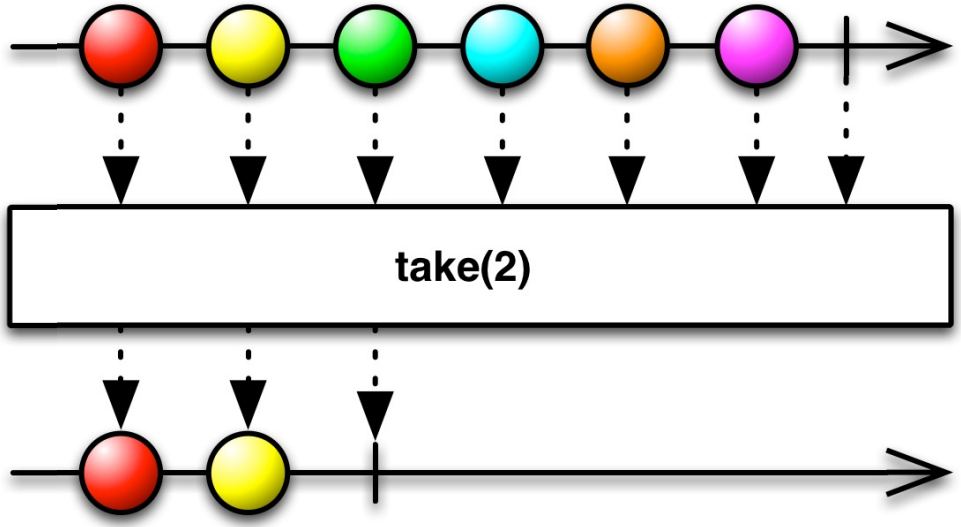
Generate an infinite series of integers periodically in a background thread

```
Observable
   .interval(sSLEEP.toMillis(),
             MILLISECONDS)
   ...
   .take(sMAX_ITERATIONS)
```

See previous discussion of the Observable.interval() method

Key Suppressing Operators in the Observable Class

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



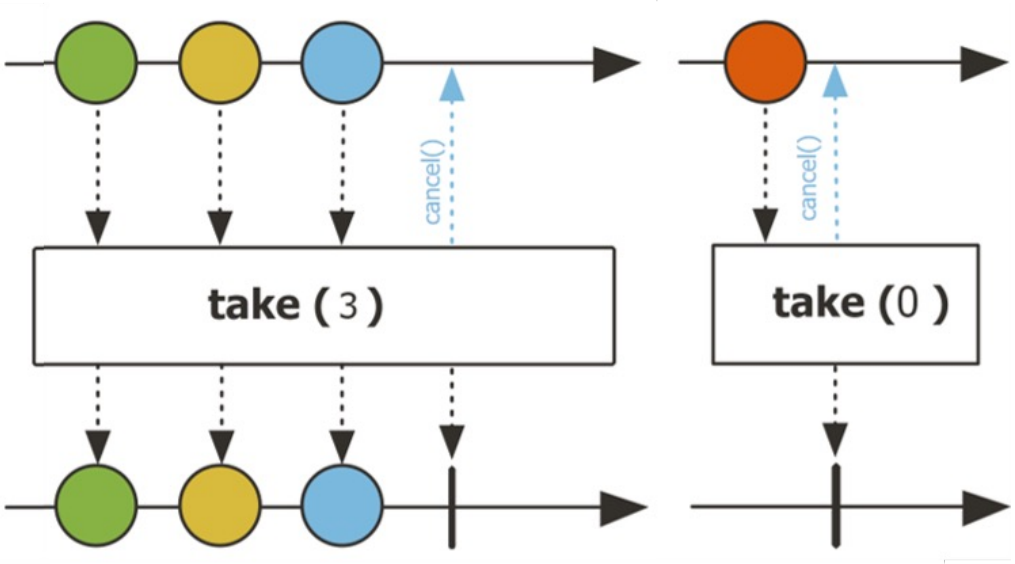

```
Observable
   .interval(sSLEEP.toMillis(),
                MILLISECONDS)
   ...
   .take(sMAX_ITERATIONS)
```

Stop emitting after sMAX_ITERATIONS

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

Key Suppressing Operators in the Observable Class

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




```
Flux
   .interval
      (sSLEEP_DURATION)
   ...
   .take(sMAX_ITERATIONS)
   ...
```

Only process sMAX_ITERATIONS # of emitted values from interval()

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

Key Suppressing Operators in the Observable Class

- The take() operator
  - Take only the first N values from this Observable, if available
  - Used to limit otherwise "infinite" streams
  - Project Reactor's Flux.take() operator 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<Long> oddNumbers = Stream
   .iterate(1L, l -> l + 1)
   .filter(n -> (n & 1) != 0)
   .limit(100)
   .collect(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 Observable Class

- The ignoreElements() operator
  - Ignores all items emitted by the current Observable

```
Completable
ignoreElements()
```

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

Key Suppressing Operators in the Observable Class

- The ignoreElements() operator
  - Ignores all items emitted by the current Observable
    - It only calls onComplete() or onError()
      - But not onNext()!

```
Completable
ignoreElements()
```

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

Key Suppressing Operators in the Observable Class

- The ignoreElements() operator
  - Ignores all items emitted by the current Observable
    - It only calls onComplete() or onError()
  - Returns a new Completable instance
    - i.e., emits no value, but only completion or error

Completable

ignoreElements()

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.

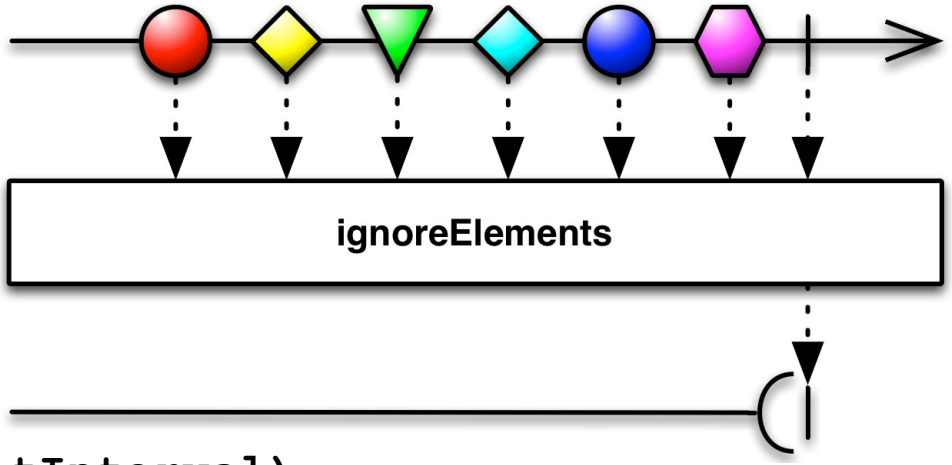
`Completable` behaves similarly to `Observable` except that it can only emit either a completion or error signal (there is no `onNext` or `onSuccess` as with the other reactive types).

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:

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

Key Suppressing Operators in the Observable Class

- The ignoreElements() operator
  - Ignores all items emitted by the current Observable
  - This "data-suppressing" operator ignores its payload



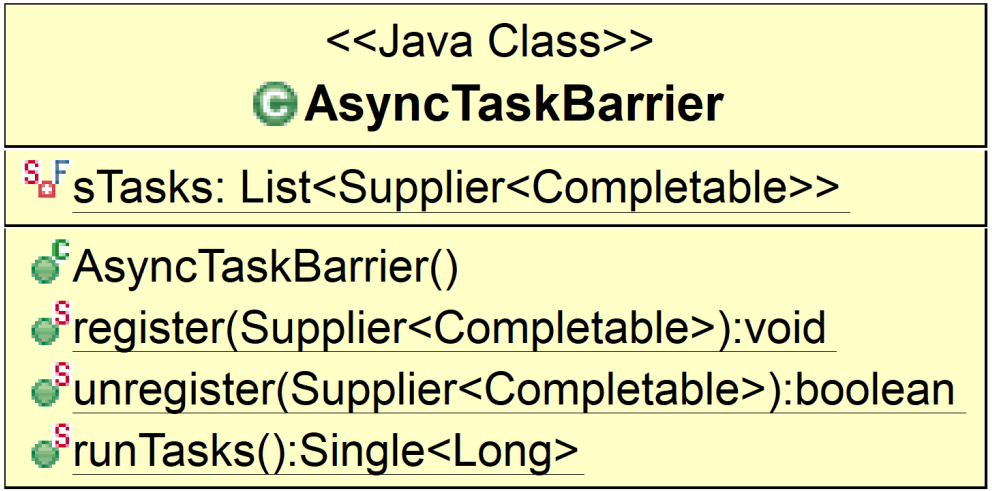

```
return Observable
        .create(ObservableEx::emitInterval)
        .map(bigInteger -> ObservableEx
                .checkIfPrime(bigInteger, sb))
        .doOnComplete(() -> BigFractionUtils
                        .display(sb.toString()))
        .ignoreElements();
```

Indicate an async operation completed

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

Key Suppressing Operators in the Observable Class

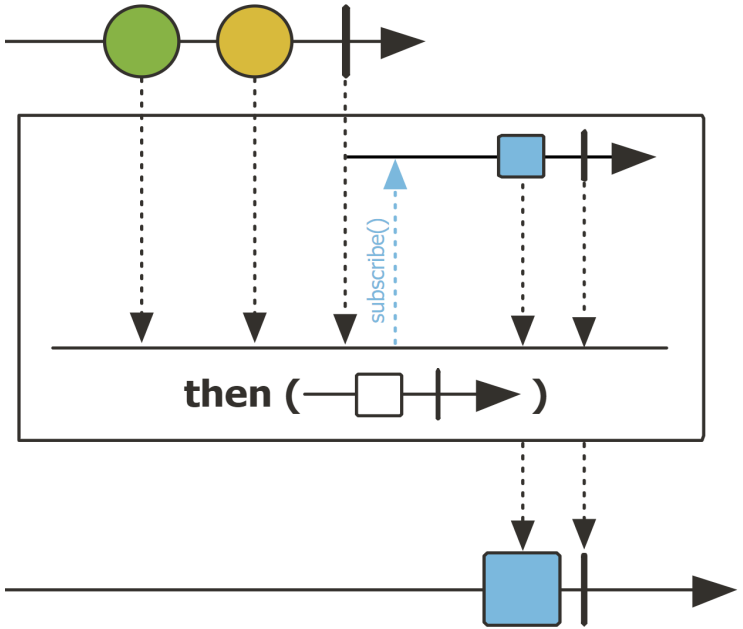
- The ignoreElements() operator
  - Ignores all items emitted by the current Observable
  - This "data-suppressing" operator ignores its payload
    - Used by the AsyncTaskBarrier framework to determine when an async task completes



See Reactive/Observable/ex2/src/main/java/utils/AsyncTaskBarrier.java

Key Suppressing Operators in the Observable Class

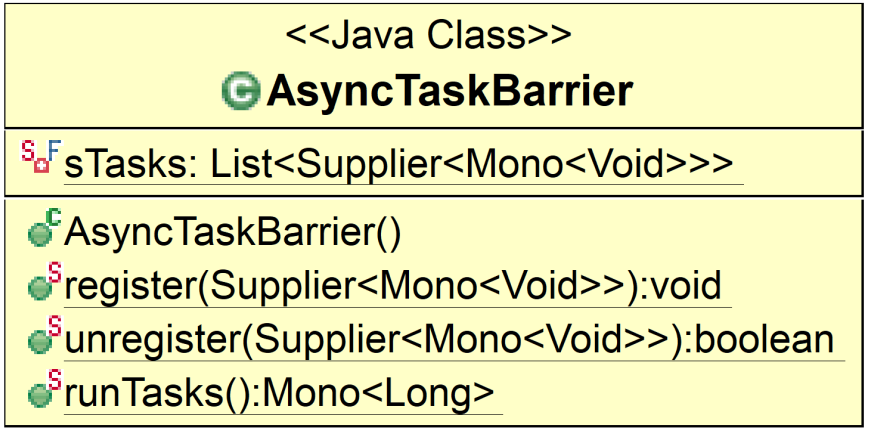
- The ignoreElements() operator
  - Ignores all items emitted by the current Observable
  - This "data-suppressing" operator ignores its payload
  - Project Reactor doesn't really have an equivalent, though its then() operator can be used in a similar way



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

Key Suppressing Operators in the Observable Class

- The ignoreElements() operator
  - Ignores all items emitted by the current Observable
  - This "data-suppressing" operator ignores its payload
  - Project Reactor doesn't really have an equivalent, though its then() operator can be used in a similar way
    - Also used by the AsyncTaskBarrier to determine when an async task completes



See Reactive/flux/ex2/src/main/java/utils/AsyncTaskBarrier.java

End of Key Suppressing Operators in the Observable Class