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 & 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

```java
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](http://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

```java
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

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

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

```java
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

```java
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](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

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

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

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

```java
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)
```
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

```java
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]( 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

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

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

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

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

- The `then()` operator: $\langle V \rangle \text{Mono}<V> \ then(Mono<V>\ other)$

- Let this Flux complete & then play signals from a provided Mono

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#then](projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#then)
Key Suppressing Operators in the Flux Class

- 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 element from this Flux & transform its completion signal into the emission & completion signal of a provided Mono
Key Suppressing Operators in the Flux Class

- The `then()` operator
  - 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

\[
<\!V\!>\quad \text{Mono}<\!V\!>\quad \text{then}(\text{Mono}<\!V\!>\quad \text{other})
\]
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

```java
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

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Completable.html](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()
End of Key Suppressing Operators in the Flux Class