

Key Factory Method Operators in the Flux Class (Part 4)

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
- Factory method operators
 - These operators create Flux streams in various ways
 - e.g., `generate()`



See en.wikipedia.org/wiki/Factory_method_pattern

Key Factory Method Operators in the Flux Class

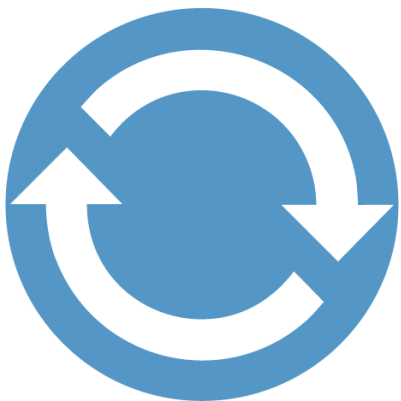
Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback

```
static <T> Flux<T> generate  
    (Consumer<SynchronousSink<T>>  
     generator)
```

Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - The Consumer param is called in a loop after a downstream Subscriber has subscribed



```
static <T> Flux<T> generate  
(Consumer<SynchronousSink<T>>  
generator)
```

Interface Consumer<T>

Type Parameters:

T - the type of the input to the operation

All Known Subinterfaces:

Stream.Builder<T>

Functional Interface:

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

Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - The Consumer param is called in a loop after a downstream Subscriber has subscribed
 - The callback should call next(), error(), or complete() on a SynchronousSink to signal a value or a terminal event

```
static <T> Flux<T> generate  
(Consumer<SynchronousSink<T>>  
generator)
```

```
public interface SynchronousSink<T>
```

Interface to produce synchronously "one signal" to an underlying Subscriber.

At most one `next(T)` call and/or one `complete()` or `error(Throwable)` should be called per invocation of the generator function.

Calling a `SynchronousSink` outside of a generator consumer or function, e.g. using an async callback, is forbidden. You can `FluxSink` or `MonoSink` based generators for these situations.

Method Summary

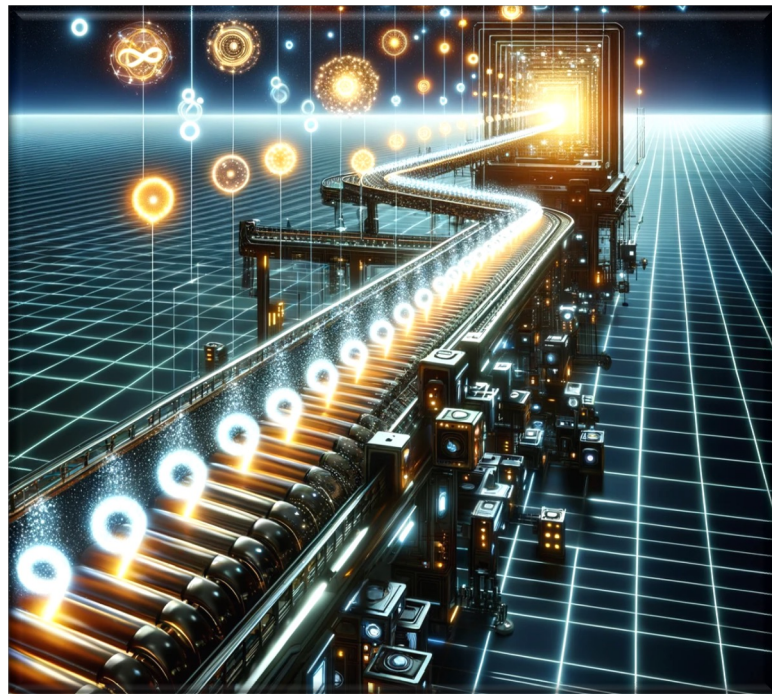
All Methods Instance Methods Abstract Methods

Modifier and Type	Method and Description
void	<code>complete()</code>
Context	<code>currentContext()</code> Return the current subscriber <code>Context</code> .
void	<code>error(Throwable e)</code>
void	<code>next(T t)</code> Try emitting, might throw an unchecked exception.

Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - The Consumer param is called in a loop after a downstream Subscriber has subscribed
 - A new Flux instance is returned that emits the events from the generator

```
static <T> Flux<T> generate  
(Consumer<SynchronousSink<T>>  
generator)
```



Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - The Consumer param is called in a loop after a downstream Subscriber has subscribed
 - The new Flux instance is returned
 - This Flux is “cold,” which only emits item upon subscription

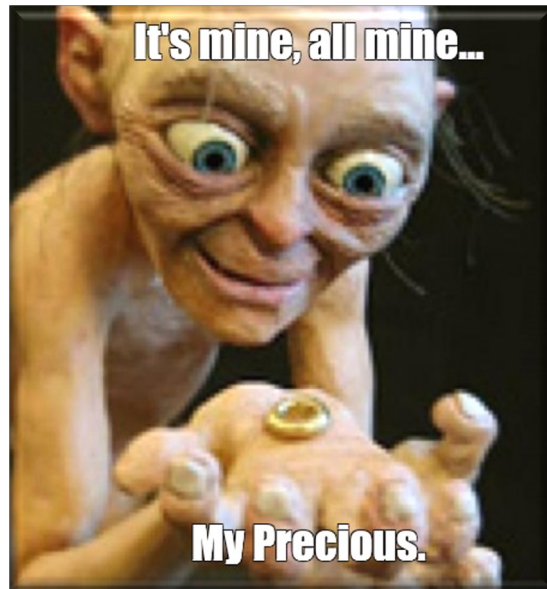
```
static <T> Flux<T> generate  
(Consumer<SynchronousSink<T>>  
generator)
```



Key Factory Method Operators in the Flux Class

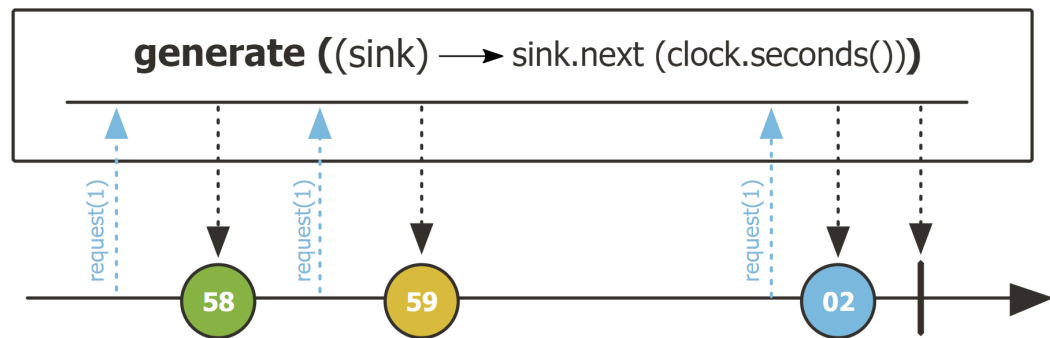
- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - The Consumer param is called in a loop after a downstream Subscriber has subscribed
 - The new Flux instance is returned
 - This Flux is "cold," which only emits item upon subscription
 - Each subscriber thus has its own set of items emitted to it

```
static <T> Flux<T> generate  
(Consumer<SynchronousSink<T>>  
generator)
```



Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - It is only allowed to generate one event at a time, which supports backpressure



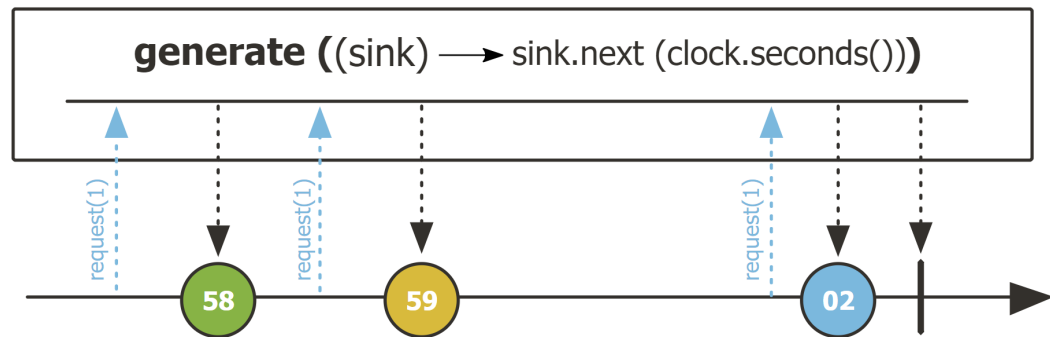
Flux

```
.generate((SynchronousSink<BigFraction> sink) -> sink  
    .next(BigFractionUtils  
        .makeBigFraction(sRANDOM,  
            false)))
```

...

Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - It is only allowed to generate one event at a time, which supports backpressure



Flux

```
.generate( (SynchronousSink<BigFraction> sink) -> sink  
    .next( BigFractionUtils  
        .makeBigFraction( sRANDOM,  
                          false)))
```

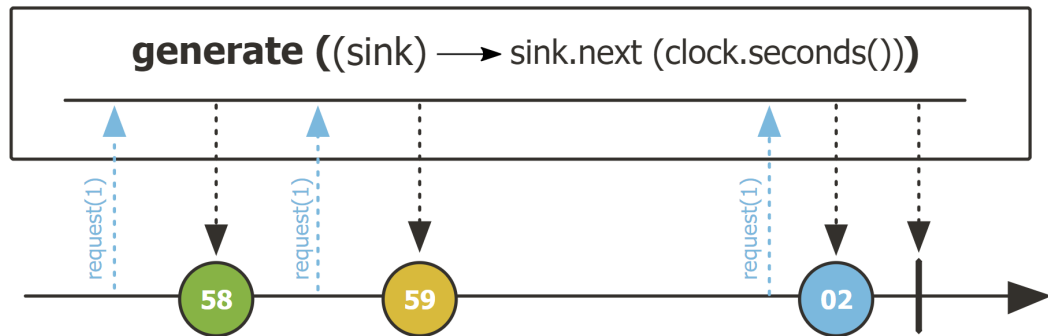
...

Generate an infinite stream of random unreduced big fractions

See [Reactive/flux/ex3/src/main/java/FluxEx.java](https://github.com/reactive/reactive-streams-examples/blob/master/reactive-streams-examples/src/main/java/FluxEx.java)

Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - It is only allowed to generate one event at a time, which supports backpressure



Flux

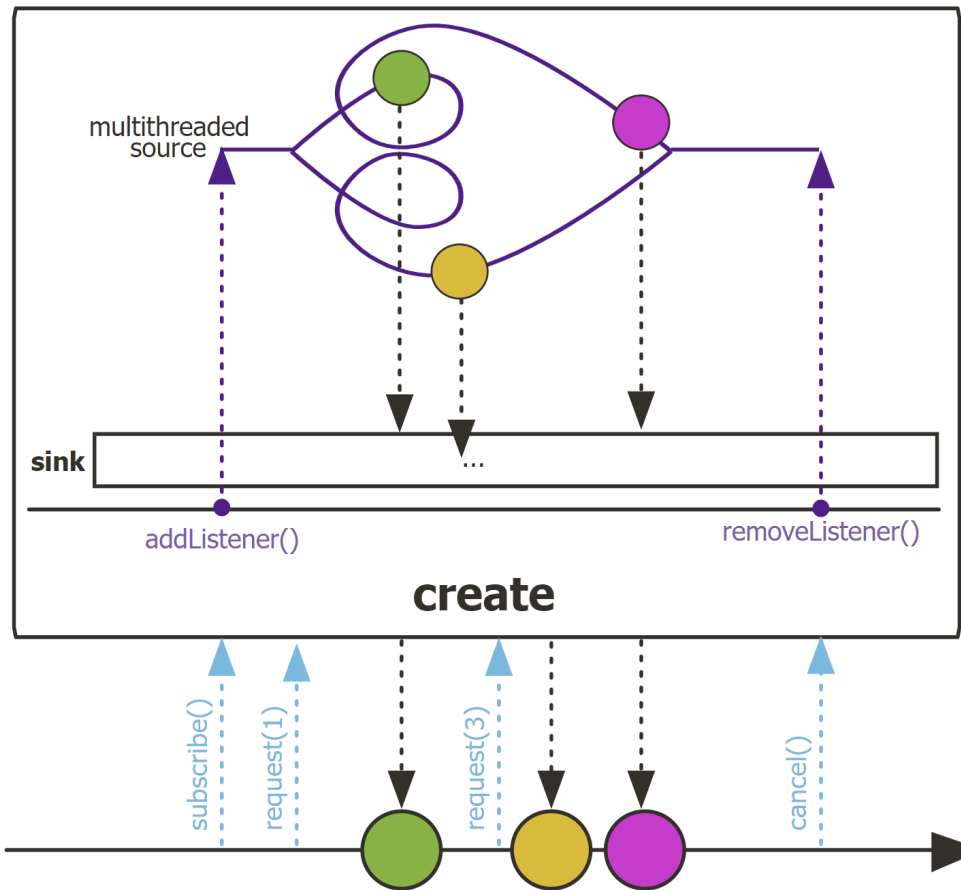
```
.generate((SynchronousSink<BigFraction> sink) -> sink  
    .next(BigFractionUtils  
        .makeBigFraction(sRANDOM,  
            false)))  
  
.take(sMAX_FRACTIONS)  
...
```

Can be used with take() to limit the number of elements generated

See earlier lesson on "Key Suppressing Operators in the Flux Class"

Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - It is only allowed to generate one event at a time, which supports backpressure
 - In contrast, the one-param create() operator produces events whenever it wants



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

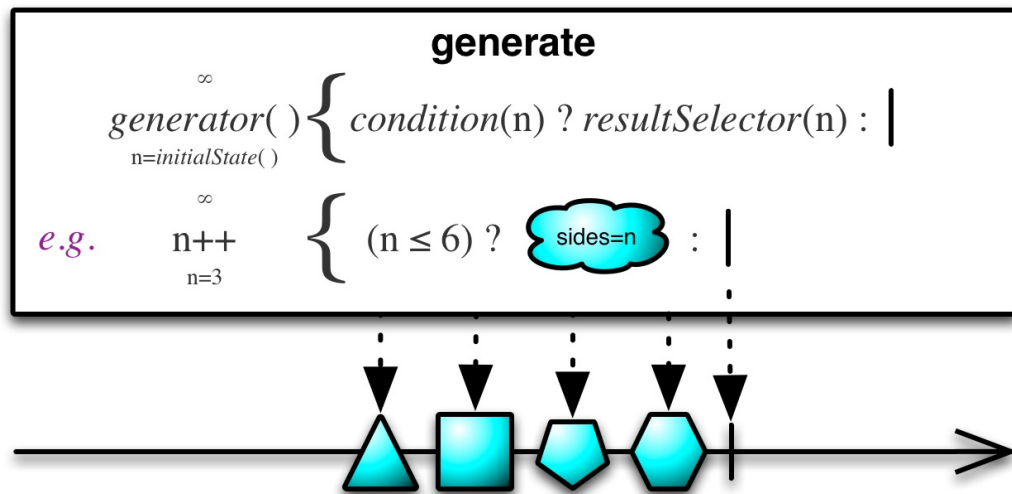
Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - It is only allowed to generate one event at a time, which supports backpressure
 - In contrast, the one-param create() operator produces events whenever it wants
 - i.e., it ignores backpressure



Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - It is only allowed to generate one event at a time, which supports backpressure
 - RxJava's Observable.generate() works in a similar way



Observable

```
.generate((Emitter<BigFraction> emit) -> emit  
.onNext(BigFractionUtils  
.makeBigFraction(sRANDOM,  
false))) ...
```

Generate a stream of random, large, & unreduced big fractions

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

Key Factory Method Operators in the Flux Class

- The generate() operator
 - Create a Flux by generating signals 1-by-1 via a callback
 - It is only allowed to generate one event at a time, which supports backpressure
 - RxJava's Observable.generate() works the same
 - Similar to Stream.generate() in Java Streams

```
generate  
  
static <T> Stream<T> generate(Supplier<T> s)  
  
Returns an infinite sequential unordered stream where each element is generated by the provided Supplier. This is suitable for generating constant streams, streams of random elements, etc.  
  
Type Parameters:  
T - the type of stream elements  
  
Parameters:  
s - the Supplier of generated elements  
  
Returns:  
a new infinite sequential unordered Stream
```

Stream

```
.generate ( () -> BigFractionUtils  
                .makeBigFraction(new Random(),  
                                false) )
```

Generate a stream of random, large, & unreduced big fractions

End of Key Factory Method Operators in the Flux Class (Part 4)