Key Factory Method Operators in the Flux 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 Flux operators
  - Concurrency & scheduler operators
  - Factory method operators
    - These operators create Flux streams in various ways
      - e.g., create(), range(), & interval()

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 create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously

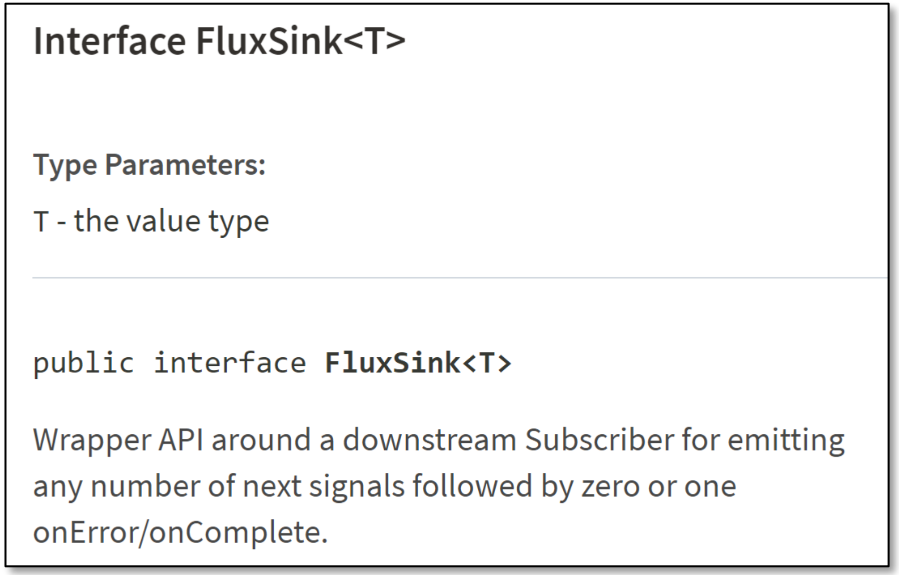
```
static <T> Flux<T> create
   (Consumer<? super FluxSink<T>>
    emitter)
```

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

Key Factory Method Operators in the Flux Class

- The create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously
    - The param emits any # of next() signals followed by zero or one error() or complete() signals

```
static <T> Flux<T> create
   (Consumer<? super FluxSink<T>>
    emitter)
```



Interface FluxSink<T>

Type Parameters:

T - the value type

```
public interface FluxSink<T>
```

Wrapper API around a downstream Subscriber for emitting any number of next signals followed by zero or one onError/onComplete.

See projectreactor.io/docs/core/release/api/reactor/core/publisher/FluxSink.html

Key Factory Method Operators in the Flux Class

- The create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously
    - The param emits any # of next() signals followed by zero or one error() or complete() signals
      - Supports more dynamic use cases than the Flux just() & fromIterable() operators

```
static <T> Flux<T> create
   (Consumer<? super FluxSink<T>>
    emitter)
```

See earlier lesson on "*Key Factory Method Operators in the Flux Class (Part 1)*"

Key Factory Method Operators in the Flux Class

- The create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously
    - The param emits any # of next() signals followed by zero or one error() or complete() signals
    - Returns a Flux that emits all the elements generated by the FluxSink

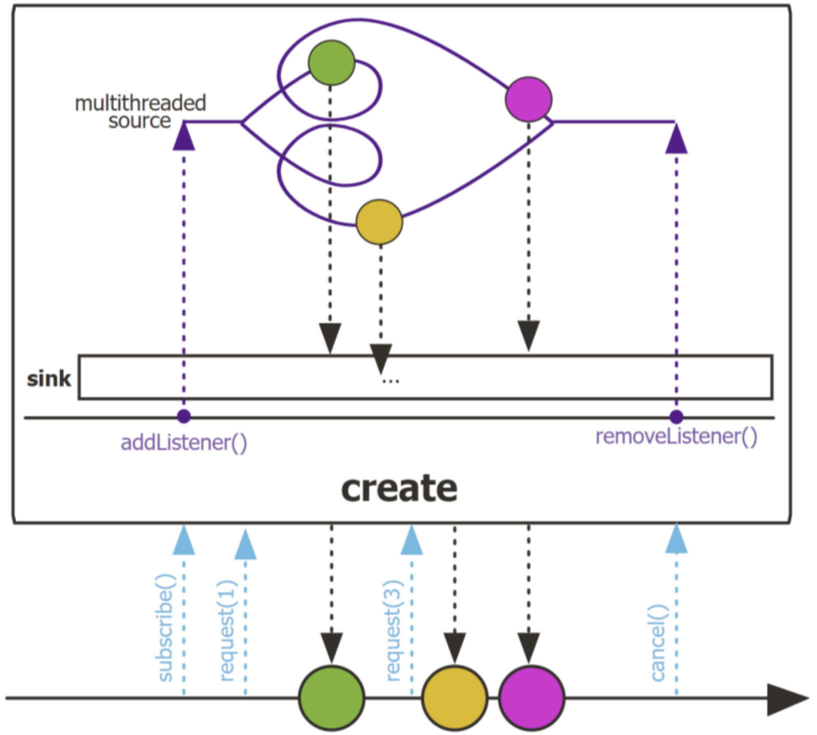
```
static <T> Flux<T> create
   (Consumer<? super FluxSink<T>>
    emitter)
```

Key Factory Method Operators in the Flux Class

- The create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously

```
static <T> Flux<T> generate
           (Supplier<T> supplier,
            long count) {
  return Flux.create(sink -> {
    for(int i = 0; i < count; ++i)
      sink.next(supplier.get());

    sink.complete();
  });
}
```

Synchronously generate 'count' instances of what's returned by supplier.get()

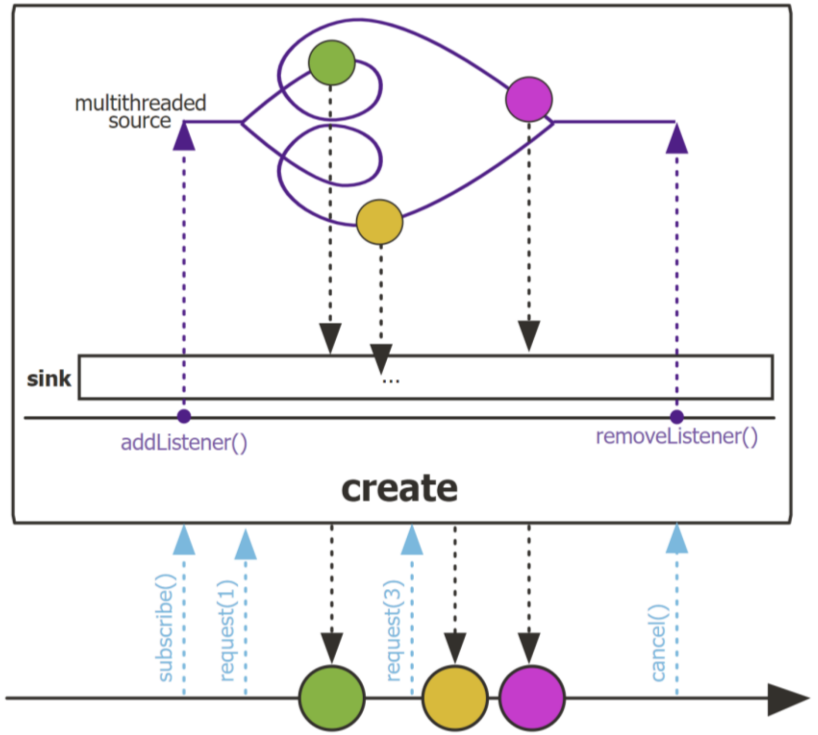


See Reactive/Flux/ex1/src/main/java/utils/ReactorUtils.java

Key Factory Method Operators in the Flux Class

- The create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously

```
static <T> Flux<T> generate
             (Supplier<T> supplier,
              long count) {
  return Flux.create(sink -> {
    for(int i = 0; i < count; ++i)
      sink.next(supplier.get()));

    sink.complete();
  });
}
```



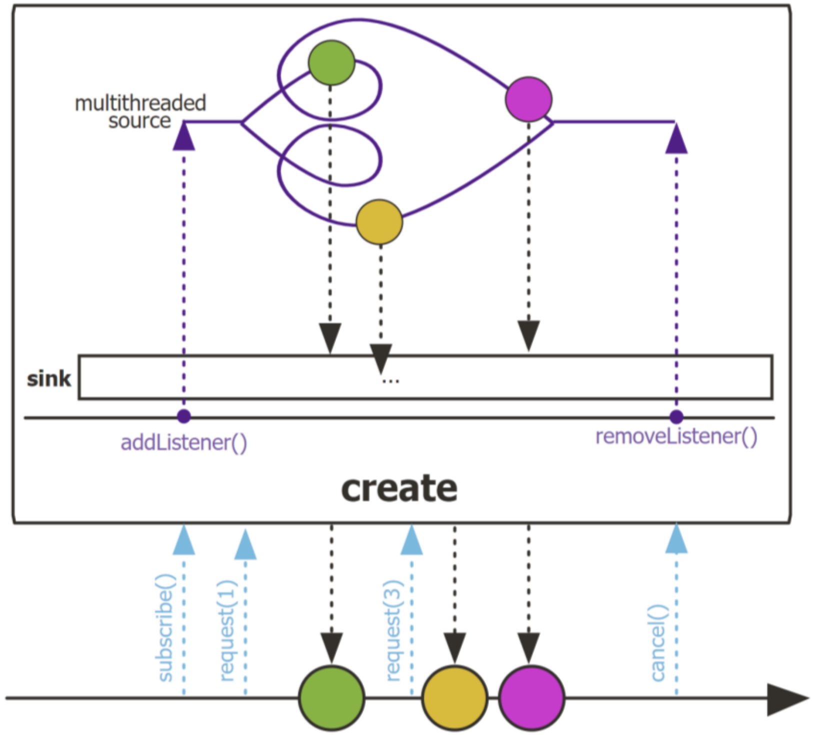
Generate the next element & emit it

Key Factory Method Operators in the Flux Class

- The create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously

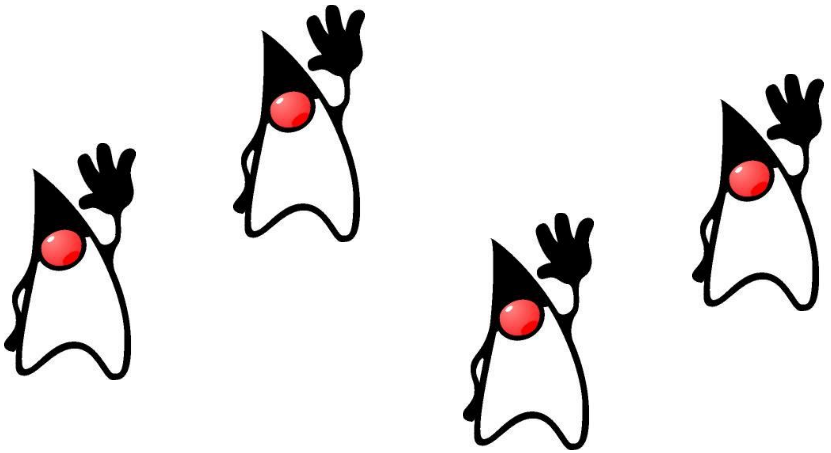
```
static <T> Flux<T> generate
          (Supplier<T> supplier,
           long count) {
  return Flux.create(sink -> {
    for(int i = 0; i < count; ++i)
      sink.next(supplier.get());

    sink.complete();
  });
}
```

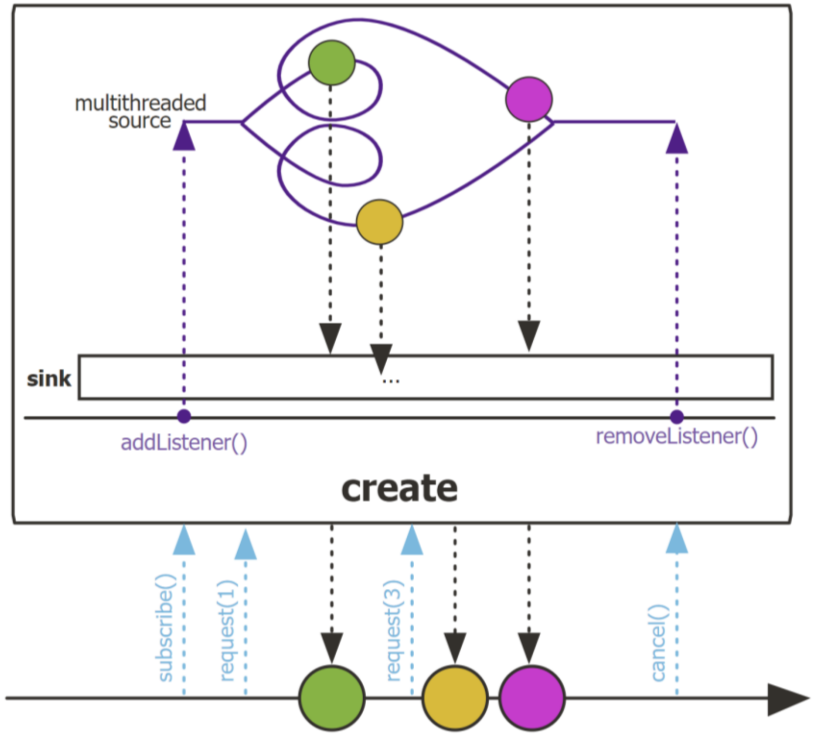
Indicate the generator is finished



Key Factory Method Operators in the Flux Class

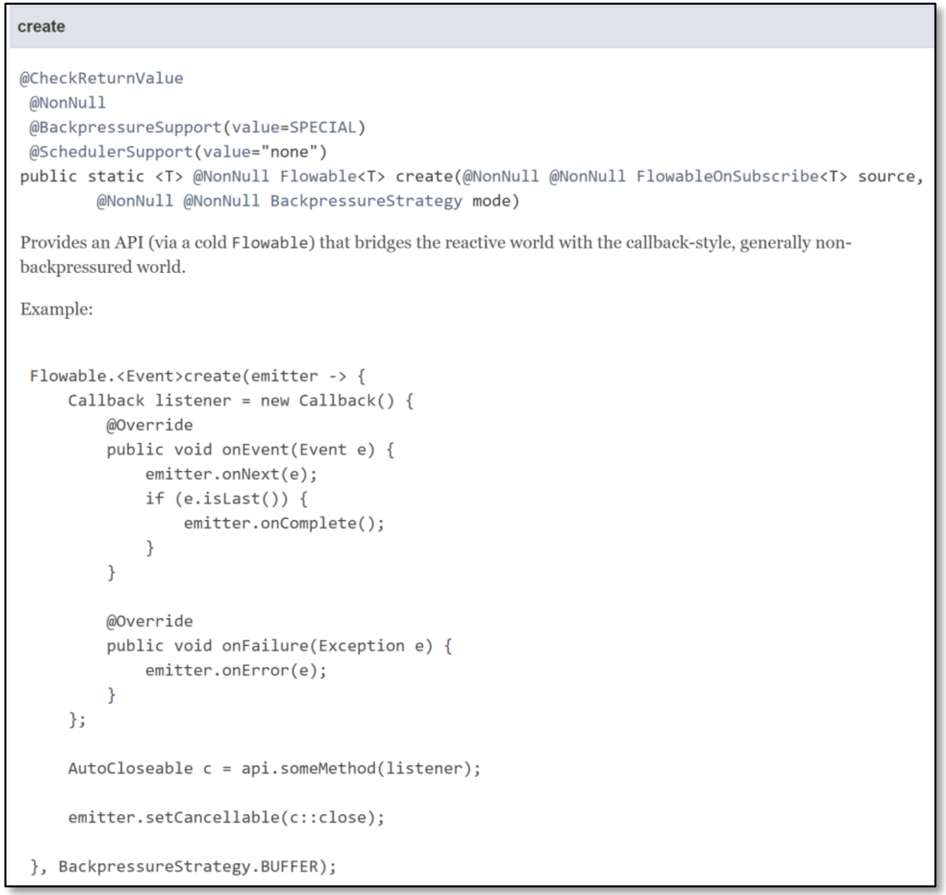
- The create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously
  - Elements can be emitted from one or more threads





Key Factory Method Operators in the Flux Class

- The create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously
  - Elements can be emitted from one or more threads
  - RxJava's Flowable.create() works in a similar way
    - However, the data types passed to create() differ



create

```
@CheckReturnValue
 @NonNull
 @BackpressureSupport(value=SPECIAL)
 @SchedulerSupport(value="none")
public static <T> @NonNull Flowable<T> create(@NonNull @NonNull FlowableOnSubscribe<T> source,
        @NonNull @NonNull BackpressureStrategy mode)
```

Provides an API (via a cold Flowable) that bridges the reactive world with the callback-style, generally non-backpressured world.

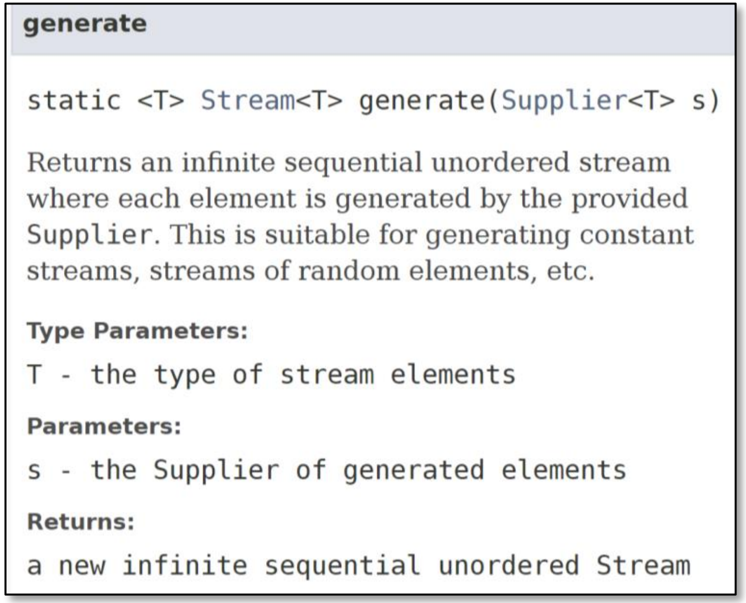
Example:

```
 Flowable.<Event>create(emitter -> {
     Callback listener = new Callback() {
         @Override
         public void onEvent(Event e) {
             emitter.onNext(e);
             if (e.isLast()) {
                 emitter.onComplete();
             }
         }

         @Override
         public void onFailure(Exception e) {
             emitter.onError(e);
         }
     };

     AutoCloseable c = api.someMethod(listener);

     emitter.setCancellable(c::close);

 }, BackpressureStrategy.BUFFER);
```

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Flowable.html#create

Key Factory Method Operators in the Flux Class

- The create() operator
  - Create a Flux capable of emitting multiple elements synchronously or asynchronously
  - Elements can be emitted from one or more threads
  - RxJava's Flowable.create() works in a similar way
  - Similar to the generate() method 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

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

Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)

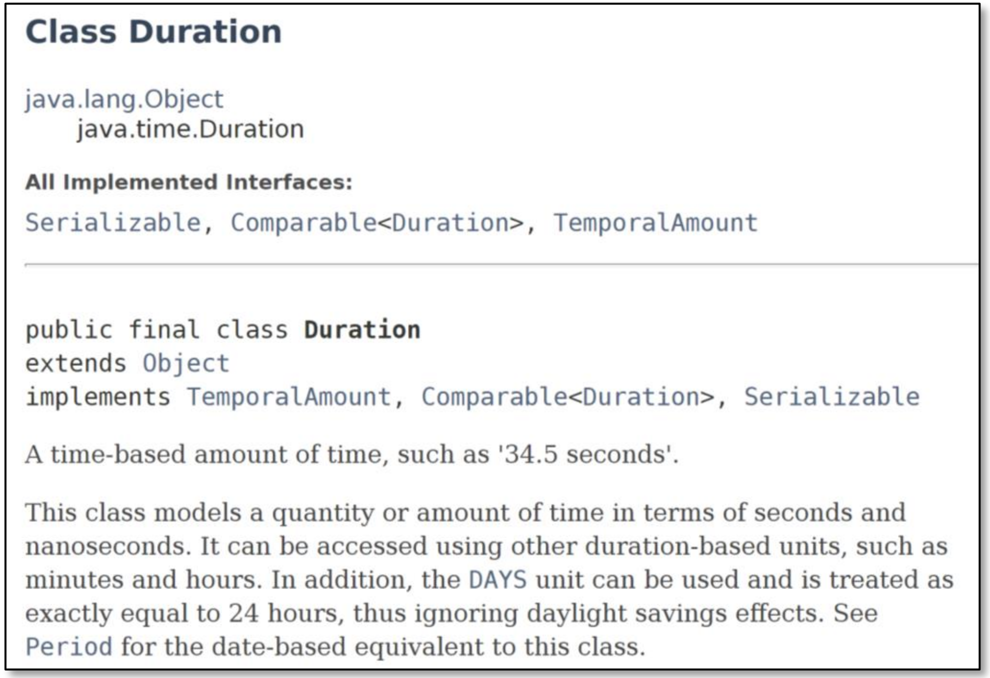
```
static Flux<Long> interval
  (Duration period)
```

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

Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)
    - The param indicates when to increment a value at the specified time interval

```
static Flux<Long> interval
   (Duration period)
```



Class Duration

java.lang.Object
    java.time.Duration

All Implemented Interfaces:

`Serializable, Comparable<Duration>, TemporalAmount`

```
public final class Duration
extends Object
implements TemporalAmount, Comparable<Duration>, Serializable
```

A time-based amount of time, such as '34.5 seconds'.

This class models a quantity or amount of time in terms of seconds and nanoseconds. It can be accessed using other duration-based units, such as minutes and hours. In addition, the `DAYS` unit can be used and is treated as exactly equal to 24 hours, thus ignoring daylight savings effects. See `Period` for the date-based equivalent to this class.

See docs.oracle.com/javase/8/docs/api/java/time/Duration.html

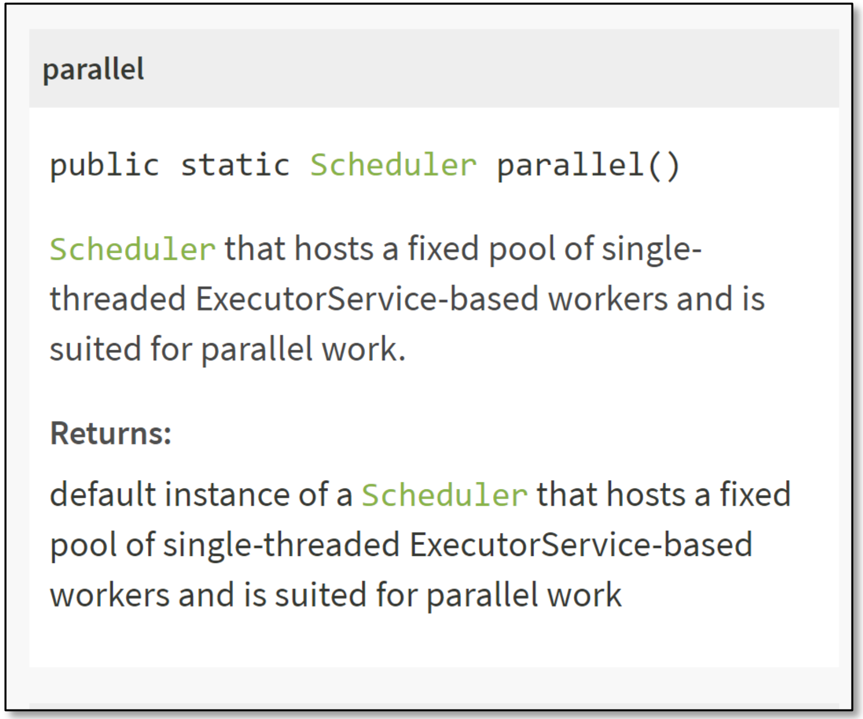
Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)
    - The param indicates when to increment a value at the specified time interval
    - Returns a new Flux emitting increasing #'s at regular intervals

```
static Flux<Long> interval
  (Duration period)
```

Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)
  - Emits values on the Schedulers .parallel() Scheduler



parallel

```
public static Scheduler parallel()
```

Scheduler that hosts a fixed pool of single-threaded ExecutorService-based workers and is suited for parallel work.

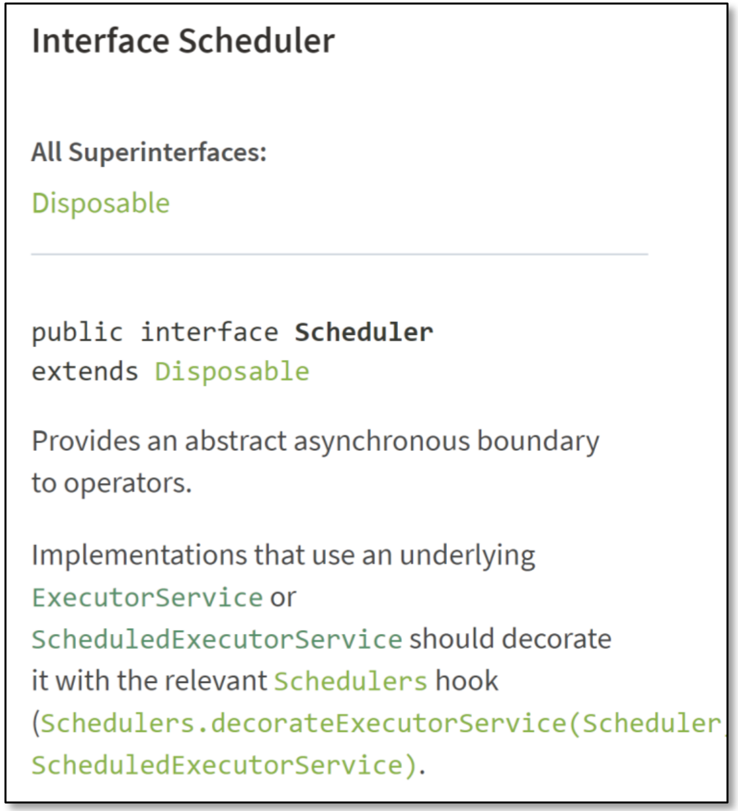
Returns:

default instance of a Scheduler that hosts a fixed pool of single-threaded ExecutorService-based workers and is suited for parallel work

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#parallel

Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)
  - Emits values on the Schedulers .parallel() Scheduler
    - Other overloaded interval() methods can designate the Scheduler



Interface Scheduler

All Superinterfaces:

Disposable

```
public interface Scheduler
extends Disposable
```

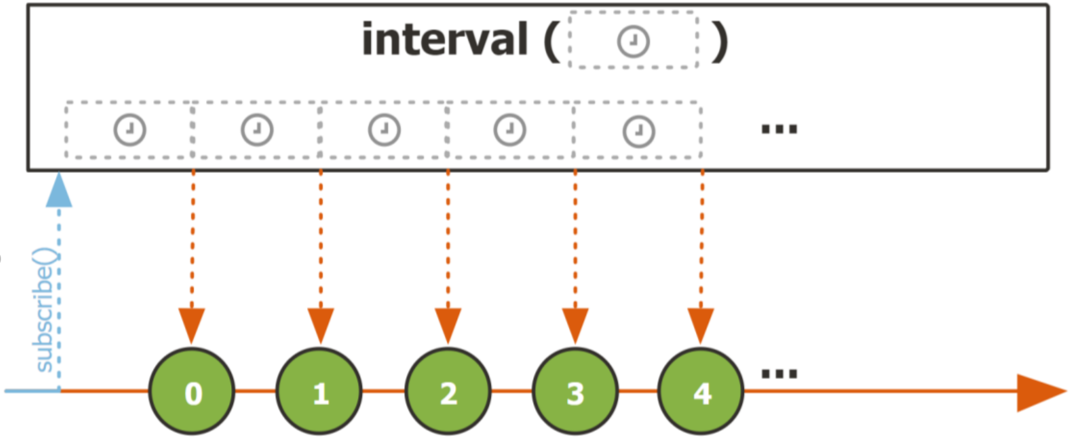
Provides an abstract asynchronous boundary to operators.

Implementations that use an underlying `ExecutorService` or `ScheduledExecutorService` should decorate it with the relevant `Schedulers` hook (`Schedulers.decorateExecutorService(Scheduler, ScheduledExecutorService)`.

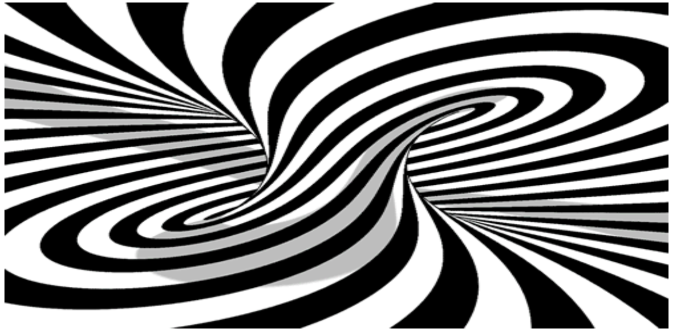
See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Scheduler.html

Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)
  - Emits values on the Schedulers .parallel() Scheduler
  - In normal conditions, the Flux will never complete



```
...
Flux
   .interval(Duration.ofMillis(500))
   ...
```

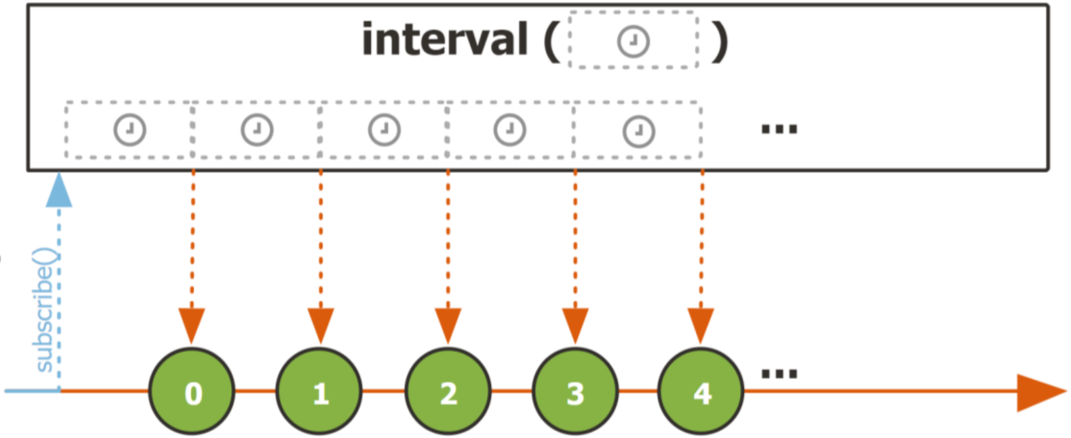


Generate a stream of longs every .5 seconds in a background thread

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

Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)
  - Emits values on the Schedulers .parallel() Scheduler
  - In normal conditions, the Flux will never complete



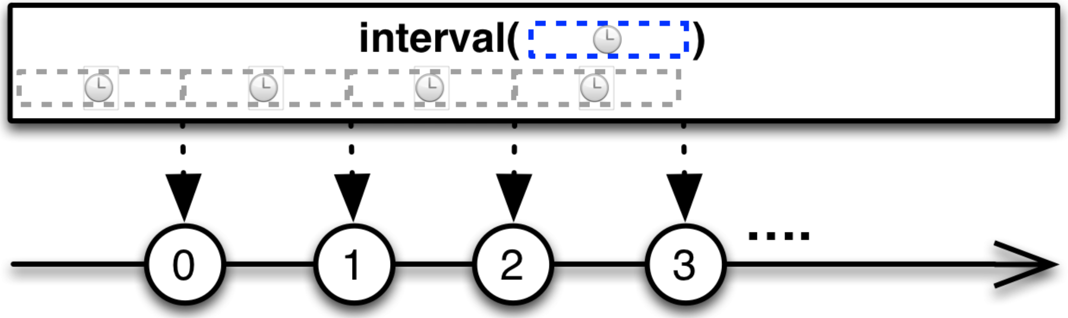
```
...
Flux
   .interval(Duration.ofMillis(500))
   ...
   .take(sMAX_ITERATIONS)
   ...
```

Use take() to only process sMAX_ITERATIONS # of emitted values from interval()

See upcoming discussion of the Flux.take() method

Key Factory Method Operators in the Flux Class

- The interval() operator
  - Create a Flux that emits long values starting with zero (0)
  - Emits values on the Schedulers .parallel() Scheduler
  - In normal conditions, the Flux will never complete
  - RxJava's Observable.interval() works the same



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

Use take() to only process sMAX_ITERATIONS # of emitted values from interval()

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

Key Factory Method Operators in the Flux Class

- The range() operator
  - Build a Flux that will only emit a sequence of ‘count’ incrementing integers, starting from ‘start’

```
static Flux<Integer> range
   (int start, int count)
```

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

Key Factory Method Operators in the Flux Class

- The range() operator
  - Build a Flux that will only emit a sequence of 'count' incrementing integers, starting from 'start'
    - Emits integers between `start' & `start + count' & then completes

```
static Flux<Integer> range
   (int start, int count)
```

Key Factory Method Operators in the Flux Class

- The range() operator
  - Build a Flux that will only emit a sequence of 'count' incrementing integers, starting from 'start'
    - Emits integers between `start' & `start + count' & then completes

```
static Flux<Integer> range
  (int start, int count)
```

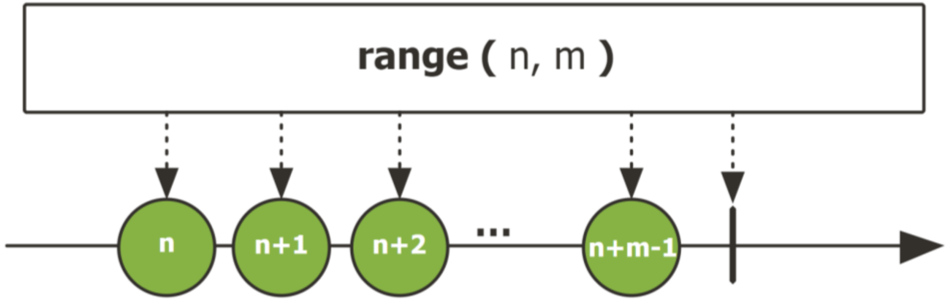
Key Factory Method Operators in the Flux Class

- The range() operator
  - Build a Flux that will only emit a sequence of 'count' incrementing integers, starting from 'start'
    - Emits integers between `start' & `start + count' & then completes
    - Returns a "ranged" Flux containing count elements

```
static Flux<Integer> range
  (int start, int count)
```

Key Factory Method Operators in the Flux Class

- The range() operator
  - Build a Flux that will only emit a sequence of 'count' incrementing integers, starting from 'start'
  - Works much like a "reactive" for loop



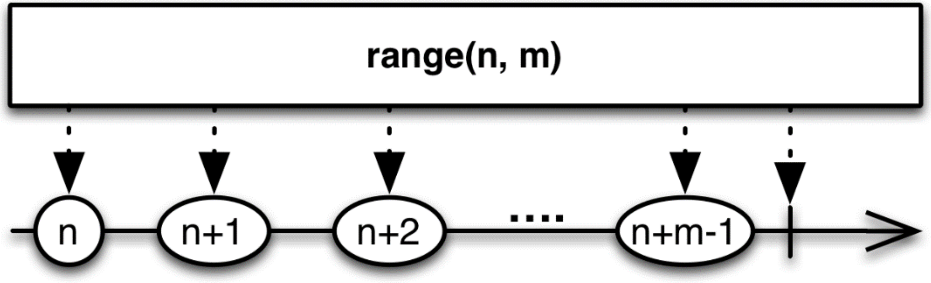
```
final int sMAX_ITERATIONS = 10;

...

Flux
  .range(1, sMAX_ITERATIONS)
  ...
```

Emit sMAX_ITERATIONS integers starting at 1

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

Key Factory Method Operators in the Flux Class

- The range() operator
  - Build a Flux that will only emit a sequence of 'count' incrementing integers, starting from 'start'
  - Works much like a "reactive" for loop
  - RxJava's Observable.range() works the same



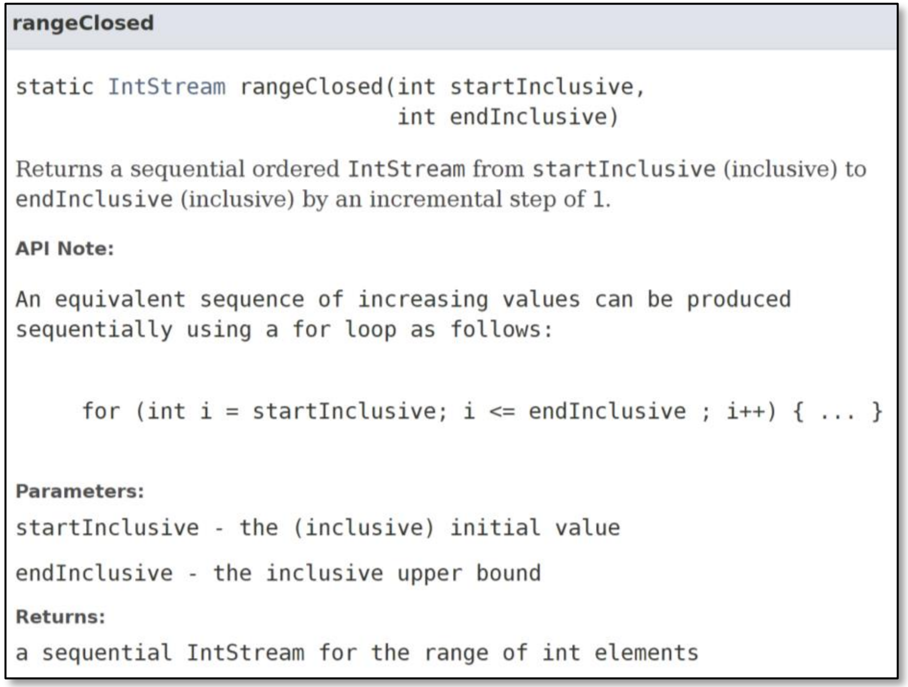
```
final int sMAX_ITERATIONS = 10;

...

Observable
  .range(1, sMAX_ITERATIONS)
  ...
```

Emit sMAX_ITERATIONS integers starting at 1

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

Key Factory Method Operators in the Flux Class

- The range() operator
  - Build a Flux that will only emit a sequence of 'count' incrementing integers, starting from 'start'
  - Works much like a "reactive" for loop
  - RxJava's Observable.range() works the same
  - Similar to IntStream.rangeClosed() in Java Streams



rangeClosed

```
static IntStream rangeClosed(int startInclusive,
                             int endInclusive)
```

Returns a sequential ordered `IntStream` from `startInclusive` (inclusive) to `endInclusive` (inclusive) by an incremental step of 1.

API Note:

An equivalent sequence of increasing values can be produced sequentially using a for loop as follows:

```
    for (int i = startInclusive; i <= endInclusive ; i++) { ... }
```

Parameters:

`startInclusive` - the (inclusive) initial value

`endInclusive` - the inclusive upper bound

Returns:

a sequential `IntStream` for the range of int elements

Emit sMAX_ITERATIONS integers starting at 1

```
IntStream.rangeClosed
   (1, sMAX_ITERATIONS)
   ...
```

See docs.oracle.com/javase/8/docs/api/java/util/stream/IntStream.html#rangeClosed

End of Key Factory Method Operators in the Flux Class (Part 2)