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

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

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#generate](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#generate)
The `generate()` operator

Create a Flux by generating signals 1-by-1 via a callback

The param is called in a loop after a downstream Subscriber has subscribed

The callback should call `next()`, `error()`, or `complete()` to signal a value or a terminal event

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

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/SynchronousSink.html](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/SynchronousSink.html)
Key Factory Method Operators in the Flux Class

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

```java
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

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

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

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

... Generate a stream of random unreduced big fractions
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, create() simply produces events whenever it wishes to do so
    - i.e., it ignores backpressure

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html#create](http://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
  - 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

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

See docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#generate
End of Key Factory Method Operators in the Flux Class (Part 3)