

# Key Factory Method Operators in the Flowable Class (Part 1)

**Douglas C. Schmidt**

**[d.schmidt@vanderbilt.edu](mailto:d.schmidt@vanderbilt.edu)**

**[www.dre.vanderbilt.edu/~schmidt](http://www.dre.vanderbilt.edu/~schmidt)**

**Professor of Computer Science**

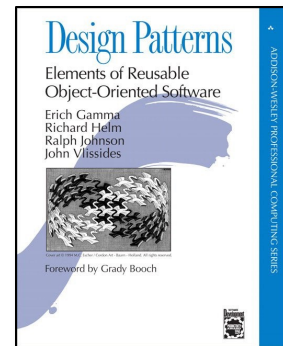
**Institute for Software  
Integrated Systems**

**Vanderbilt University  
Nashville, Tennessee, USA**



# Learning Objectives in this Part of the Lesson

- Recognize key operators defined in—or used with—Flowable
  - Factory method operators
    - These operators create Flowable streams in various ways
      - e.g., create()



See [en.wikipedia.org/wiki/Factory\\_method\\_pattern](https://en.wikipedia.org/wiki/Factory_method_pattern)

---

# Key Factory Method Operators in the Flowable Class

# Key Factory Method Operators in the Flowable Class

---

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world

```
static <T> Flowable<T> create  
(FlowableOnSubscribe<T> source,  
BackpressureStrategy mode)
```

# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
  - The FlowableOnSubscribe() subscribe() method receives an FlowableEmitter instance

```
static <T> Flowable<T> create  
(FlowableOnSubscribe<T> source,  
BackpressureStrategy mode)
```

```
@FunctionalInterface  
public interface FlowableOnSubscribe<T>
```

A functional interface that has a subscribe() method that receives a FlowableEmitter instance that allows pushing events in a backpressure-safe and cancellation-safe manner.

## Method Summary

All Methods

Instance Methods

Abstract Methods

Modifier and Type Method and Description

void	<b>subscribe</b> (@NonNull FlowableEmitter<T> emitter) Called for each Subscriber that subscribes.
------	-------------------------------------------------------------------------------------------------------

# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
  - The FlowableOnSubscribe() subscribe() method receives an FlowableEmitter instance
  - FlowableEmitter can emit events via onNext(), onError(), & onComplete()

```
static <T> Flowable<T> create  
(FlowableOnSubscribe<T> source,  
BackpressureStrategy mode)
```

## Interface ObservableEmitter<T>

### Type Parameters:

T - the value type to emit

### All Superinterfaces:

Emitter<T>

```
public interface ObservableEmitter<T>  
extends Emitter<T>
```

Abstraction over an RxJava Observer that allows associating a resource with it.

The Emitter.onNext(Object), Emitter.onError(Throwable), tryOnError(Throwable) and Emitter.onComplete() methods should be called in a sequential manner, just like the Observer's methods should be. Use the ObservableEmitter the serialize() method returns instead of the original ObservableEmitter instance provided by the generator routine if you want to ensure this. The other methods are thread-safe.

# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
  - The FlowableOnSubscribe() subscribe() method receives an FlowableEmitter instance
    - FlowableEmitter can emit events via onNext(), onError(), & onComplete()
  - Supports more dynamic use cases than the Flowable & Observable just() & fromIterable() operators

```
static <T> Flowable<T> create  
(FlowableOnSubscribe<T> source,  
BackpressureStrategy mode)
```



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

# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
  - The FlowableOnSubscribe() subscribe() method receives an FlowableEmitter instance
- Defines the backpressure mode
  - Applied if the downstream Subscriber doesn't request (fast) enough

```
static <T> Flowable<T> create  
(FlowableOnSubscribe<T> source,  
BackpressureStrategy mode)
```

```
public enum BackpressureStrategy  
extends Enum<BackpressureStrategy>
```

Represents the options for applying backpressure to a source sequence.

## Enum Constant Summary

### Enum Constants

#### Enum Constant and Description

##### **BUFFER**

Buffers *all* onNext values until the downstream consumes it.

##### **DROP**

Drops the most recent onNext value if the downstream can't keep up.

##### **ERROR**

Signals a **MissingBackpressureException** in case the downstream can't keep up.

##### **LATEST**

Keeps only the latest onNext value, overwriting any previous value if the downstream can't keep up.

##### **MISSING**

The onNext events are written without any buffering or dropping.



# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
    - The FlowableOnSubscribe() subscribe() method receives an FlowableEmitter instance
    - Defines the backpressure mode
    - Returns a 'cold' Flowable
      - A cold Flowable emits elements from the FlowableEmitter for each new Subscriber

```
static <T> Flowable<T> create  
(FlowableOnSubscribe<T> source,  
BackpressureStrategy mode)
```



# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
    - The FlowableOnSubscribe() subscribe() method receives an FlowableEmitter instance
    - Defines the backpressure mode
    - Returns a 'cold' Flowable
      - A cold Flowable emits elements from the FlowableEmitter for each new Subscriber
        - Subject to the BackpressureStrategy mode

```
static <T> Flowable<T> create  
(FlowableOnSubscribe<T> source,  
BackpressureStrategy mode)
```

Enum Constants	
Enum Constant and Description	
<b>BUFFER</b>	Buffers <i>all</i> onNext values until the downstream consumes it.
<b>DROP</b>	Drops the most recent onNext value if the downstream can't keep up.
<b>ERROR</b>	Signals a <b>MissingBackpressureException</b> in case the downstream can't keep up.
<b>LATEST</b>	Keeps only the latest onNext value, overwriting any previous value if the downstream can't keep up.
<b>MISSING</b>	The onNext events are written without any buffering or dropping.

# Key Factory Method Operators in the Flowable Class

---

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
    - The FlowableOnSubscribe() subscribe() method receives an FlowableEmitter instance
    - Defines the backpressure mode
    - Returns a 'cold' Flowable
      - A cold Flowable emits elements from the FlowableEmitter for each new Subscriber
      - A hot Flowable emits elements even if there are no Subscribers

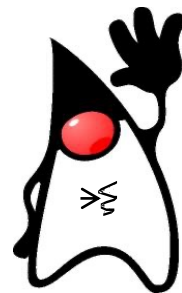
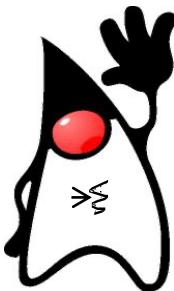
```
static <T> Flowable<T> create  
(FlowableOnSubscribe<T> source,  
BackpressureStrategy mode)
```



# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
  - Elements can be emitted from one or more threads

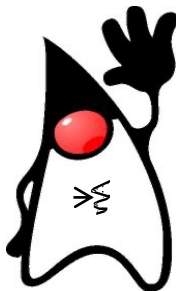
```
return Flowable
    .create(emitter -> { Flowable
        .range(1, count)
        .subscribe(____ ->
            emitter.onNext(random
                .nextInt(maxValue)),
            emitter::onError,
            emitter::onComplete);
        })
    ...
    .subscribeOn(scheduler);
```



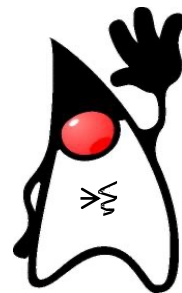
# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
  - Elements can be emitted from one or more threads

*Rapidly generate  
'count' events*



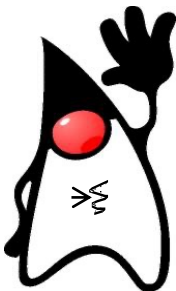
```
return Flowable
    .create(emitter -> { Flowable
        .range(1, count)
        .subscribe(____ ->
            emitter.onNext(random
                .nextInt(maxValue)),
            emitter::onError,
            emitter::onComplete);
        })
    ...
    .subscribeOn(scheduler);
```



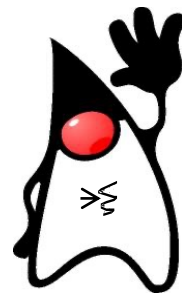
# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
  - Elements can be emitted from one or more threads

```
return Flowable
    .create(emitter -> { Flowable
        .range(1, count)
        .subscribe(____ ->
            emitter.onNext(random
                .nextInt(maxValue)),
            emitter::onError,
            emitter::onComplete);
        })
    ...
    .subscribeOn(scheduler);
```



*This emitter uses a background thread*





# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
  - Elements can be emitted from one or more threads
- Project Reactor's Flux.create() operator works in a similar way
  - However, it supports backpressure-aware Publisher(s) & Subscriber(s), as well as backpressure strategies

## Backpressure in Project reactor

You will learn about **Backpressure in the Project reactor**. Backpressure is the ability of a Consumer to signal the Producer that the rate of emission is higher than what it can handle. So using this mechanism, the Consumer gets control over the speed at which data is emitted.

If you are new to Project Reactor, read about the [Flux in reactive stream](#).

### What is Backpressure?

- Using **Backpressure**, the Subscriber controls the data flow from the Publisher.
- The Subscriber makes use of `request(n)` to request `n` number of elements at a time.

See [jstobigdata.com/java/backpressure-in-project-reactor](https://jstobigdata.com/java/backpressure-in-project-reactor)



# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
  - Elements can be emitted from one or more threads
  - Project Reactor's Flux.create() operator works in a similar way
  - Java Streams generate() method doesn't support backpressure

## 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

# Key Factory Method Operators in the Flowable Class

- The create() operator
  - Bridges the reactive world with the callback-style, non-back-pressure-aware world
  - Elements can be emitted from one or more threads
  - Project Reactor's Flux.create() operator works in a similar way
- Java Streams generate() method doesn't support backpressure
  - However, it is "pull-based" model rather than "push-based" pub/sub model, so backpressure support is not necessary

## 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



---

# End of Key Factory Method Operators in the Flowable Class (Part 1)