

Key Factory Method Operators in the Observable Class (Part 1)

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

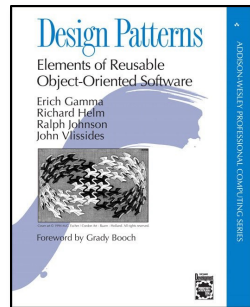
**Institute for Software
Integrated Systems**

**Vanderbilt University
Nashville, Tennessee, USA**



Learning Objectives in this Part of the Lesson

- Recognize key Observable operators
- Factory method operators
 - These operators create reactive Observable streams in various ways from non-reactive input sources
 - e.g., `just()`, `fromArray()`, `fromIterable()`, & `fromCallable()`



See en.wikipedia.org/wiki/Factory_method_pattern

Key Factory Method Operators in the Observable Class

Key Factory Method Operators in the Observable Class

- The just() operator
 - Creates an Observable that emits the given element(s) & then completes

```
static <T> Observable<T>  
    just(T... data)
```

Key Factory Method Operators in the Observable Class

- The just() operator
 - Creates an Observable that emits the given element(s) & then completes
 - The param(s) are the elements to emit, as a varargs param

```
static <T> Observable<T>  
just(T... data)
```

Key Factory Method Operators in the Observable Class

- The just() operator
 - Creates an Observable that emits the given element(s) & then completes
 - The param(s) are the elements to emit, as a varargs param
 - Returns a new Observable that's captured at "assembly time"
 - i.e., it's "eager"

```
static <T> Observable<T>  
just(T... data)
```

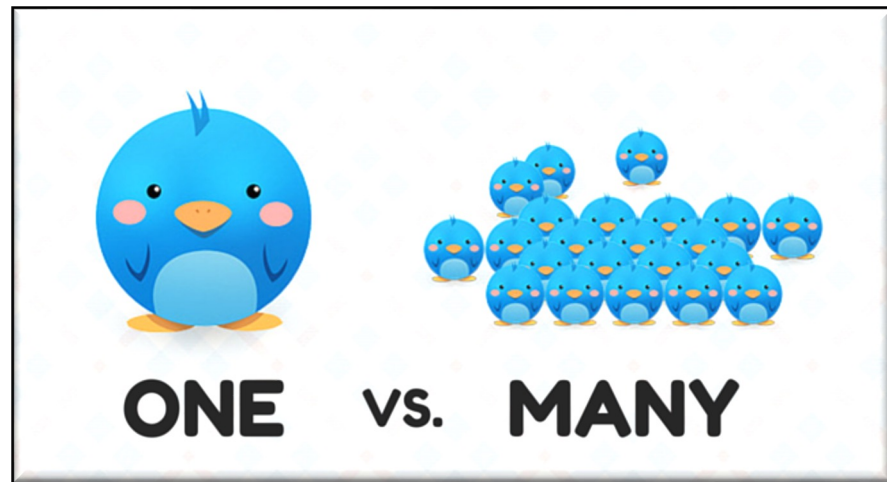


Contrast with the discussion of the Observable.fromCallable() operator later in this lesson

Key Factory Method Operators in the Observable Class

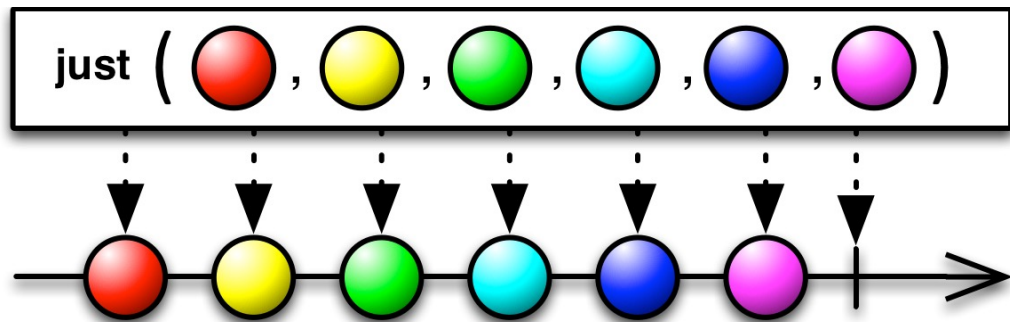
- The just() operator
 - Creates an Observable that emits the given element(s) & then completes
 - The param(s) are the elements to emit, as a varargs param
 - Returns a new Observable that's captured at "assembly time"
 - Multiple elements can be emitted, unlike the Single.just() operator

```
static <T> Observable<T>  
just(T... data)
```



Key Factory Method Operators in the Observable Class

- The just() operator
 - Creates an Observable that emits the given element(s) & then completes
 - This factory method adapts non-reactive input sources into the reactive model



Observable

*Create an Observable stream
of four BigFraction objects*

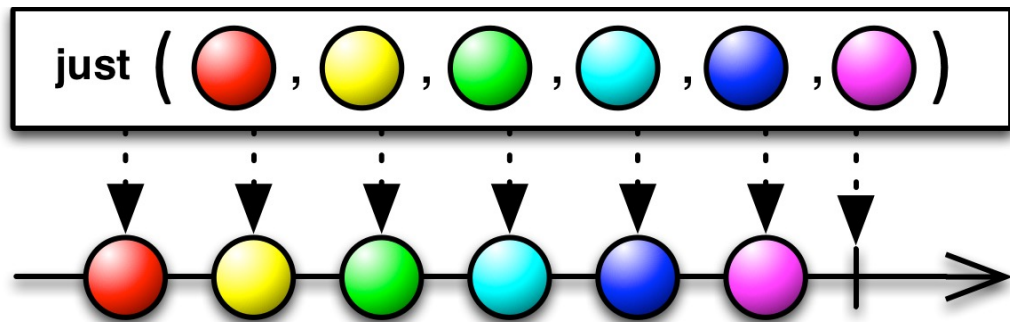
```
.just(BigFraction.valueOf(100,3),  
      BigFraction.valueOf(100,4),  
      BigFraction.valueOf(100,2),  
      BigFraction.valueOf(100,1))
```

...

See [Reactive/Observable/ex1/src/main/java/ObservableEx.java](#)

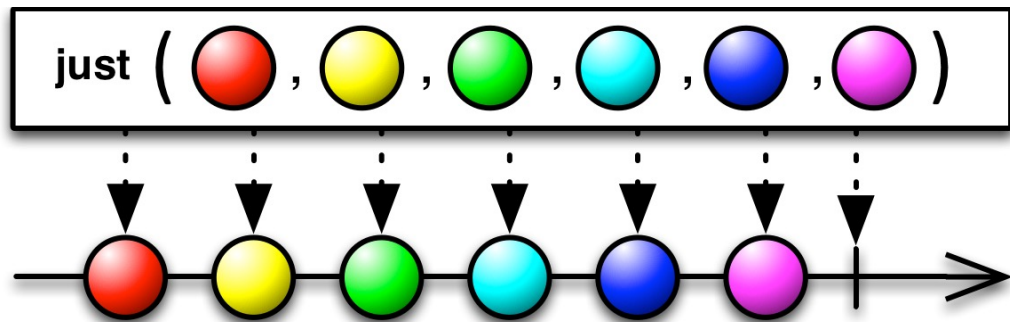
Key Factory Method Operators in the Observable Class

- The just() operator
 - Creates an Observable that emits the given element(s) & then completes
 - This factory method adapts non-reactive input sources into the reactive model
 - just() is evaluated eagerly at “assembly time”



Key Factory Method Operators in the Observable Class

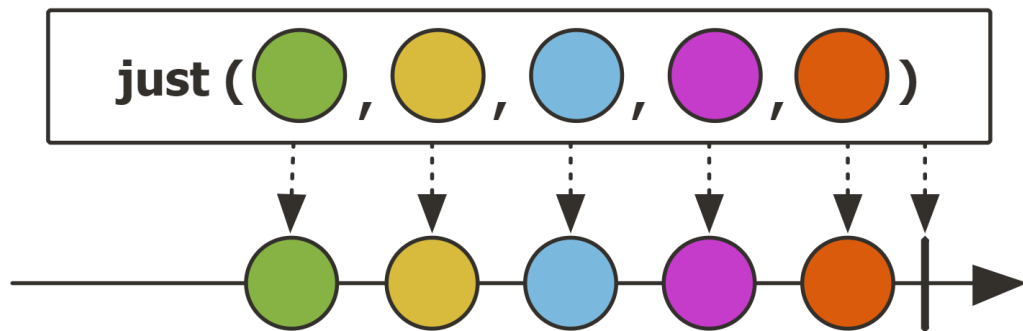
- The just() operator
 - Creates an Observable that emits the given element(s) & then completes
 - This factory method adapts non-reactive input sources into the reactive model
 - just() is evaluated eagerly at “assembly time”
 - It therefore always runs in the context of the thread where the Observable is instantiated



The `fromIterable()` & `fromArray()` factory method operators also evaluate eagerly

Key Factory Method Operators in the Observable Class

- The just() operator
 - Creates an Observable that emits the given element(s) & then completes
 - This factory method adapts non-reactive input sources into the reactive model
- Project Reactor's Flux.just() operator works the same



Flux

```
.just(BigFraction.valueOf(100,3),  
      BigFraction.valueOf(100,4),  
      BigFraction.valueOf(100,2),  
      BigFraction.valueOf(100,1))
```

...

*Create a Flux stream of
four BigFraction objects*

Key Factory Method Operators in the Observable Class

- The just() operator
 - Creates an Observable that emits the given element(s) & then completes
 - This factory method adapts non-reactive input sources into the reactive model
 - Project Reactor's Flux.just() operator works the same
 - Similar to Stream.of() factory method in Java Streams

Create a stream of 4 BigFraction objects

of

@SafeVarargs

static <T> Stream<T> of(T... values)

Returns a sequential ordered stream whose elements are the specified values.

Type Parameters:

T - the type of stream elements

Parameters:

values - the elements of the new stream

Returns:

the new stream

Stream

```
.of(BigFraction.valueOf(100,3),  
    BigFraction.valueOf(100,4),  
    BigFraction.valueOf(100,2),  
    BigFraction.valueOf(100,1))
```

...

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

Key Factory Method Operators in the Observable Class

- The `fromArray()` operator
 - Create an Observable that emits items from a Java built-in array

```
static <T> Observable<T>  
    fromArray(T[] array)
```

Key Factory Method Operators in the Observable Class

- The fromArray() operator
 - Create an Observable that emits items from a Java built-in array
 - The param provides the array to read the data from

```
static <T> Observable<T>  
    fromArray(T[] array)
```

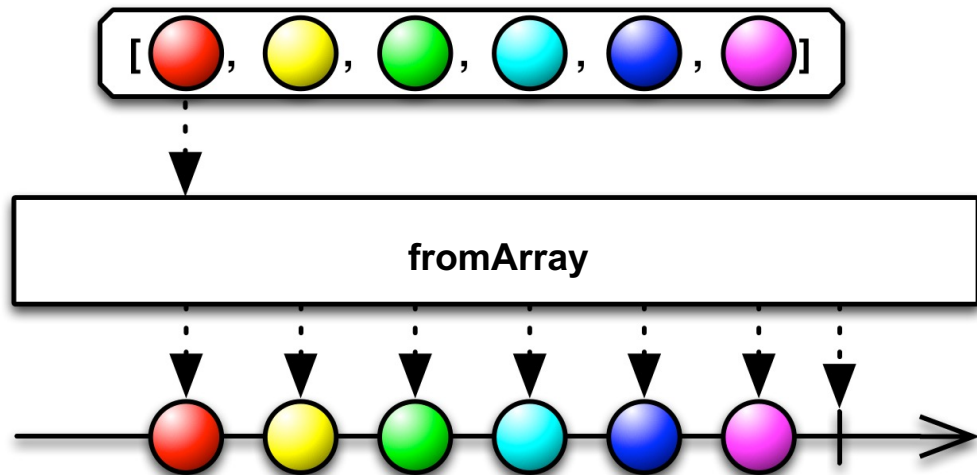
Key Factory Method Operators in the Observable Class

- The `fromArray()` operator
 - Create an Observable that emits items from a Java built-in array
 - The param provides the array to read the data from
 - The returned Observable emits the items from the array

```
static <T> Observable<T>  
    fromArray(T[] array)
```

Key Factory Method Operators in the Observable Class

- The `fromArray()` operator
 - Create an Observable that emits items from a Java built-in array
 - This factory method operator also adapts non-reactive input sources into the reactive model



```
Integer[] array =  
    {0, 1, 1, 2, 3, 5, 8, 13, 21};
```

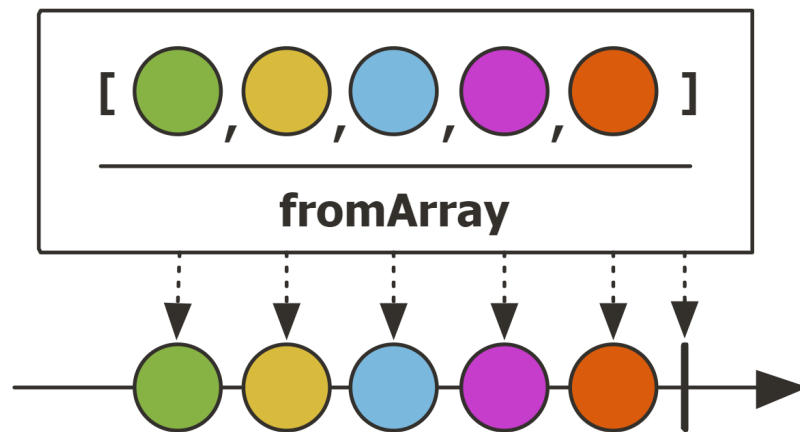
Create an Observable stream of Integer objects from a built-in array

```
Observable  
    .fromArray(array)  
    ...
```

See [Reactive/Observable/ex1/src/main/java/ObservableEx.java](#)

Key Factory Method Operators in the Observable Class

- The `fromArray()` operator
 - Create an Observable that emits items from a Java built-in array
 - This factory method operator also adapts non-reactive input sources into the reactive model
- Project Reactor's operator `Flux.fromArray()` works the same



```
Integer[] array =  
    {0, 1, 1, 2, 3, 5, 8, 13, 21};
```

Create a Flux stream of Integer objects from a Java built-in array

```
Flux  
    .fromArray(array)  
    ...
```

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

Key Factory Method Operators in the Observable Class

- The fromArray() operator
 - Create an Observable that emits items from a Java built-in array
 - This factory method operator also adapts non-reactive input sources into the reactive model
 - Project Reactor's operator Flux.fromArray() works the same
 - Similar to the of() method in Java Streams

Create a stream of Integer objects from a built-in array

of

@SafeVarargs

static <T> Stream<T> of(T... values)

Returns a sequential ordered stream whose elements are the specified values.

Type Parameters:

T - the type of stream elements

Parameters:

values - the elements of the new stream

Returns:

the new stream

```
Integer[] array =  
    {0, 1, 1, 2, 3, 5, 8, 13, 21};
```

```
Stream  
    .of(array)  
    ...
```

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

Key Factory Method Operators in the Observable Class

- The `fromArray()` operator
 - Create an Observable that emits items from a Java built-in array
 - This factory method operator also adapts non-reactive input sources into the reactive model
 - Project Reactor's operator `Flux.fromArray()` works the same
- Similar to the `of()` method in Java Streams
 - Also, similar to the `stream()` method in Java Arrays

stream

```
public static <T> Stream<T> stream(T[] array)
```

Returns a sequential `Stream` with the specified array as its source.

Type Parameters:

T - The type of the array elements

Parameters:

array - The array, assumed to be unmodified during use

Returns:

a `Stream` for the array

```
Integer[] array =  
    {0, 1, 1, 2, 3, 5, 8, 13, 21};
```

```
Arrays  
    .stream(array)  
    ...
```

*Create a stream
of Integer objects
from a built-in array*

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

Key Factory Method Operators in the Observable Class

- The fromIterable() operator
 - Create an Observable that emits the items contained in the given Iterable

```
static <T> Observable<T>  
fromIterable  
    (Iterable<? extends T> it)
```

Key Factory Method Operators in the Observable Class

- The fromIterable() operator
 - Create an Observable that emits the items contained in the given Iterable
 - The Iterable.iterator() method will be invoked at least once & at most twice for each subscriber

```
static <T> Observable<T>  
fromIterable  
(Iterable<? extends T> it)
```

Interface Iterable<T>

Type Parameters:

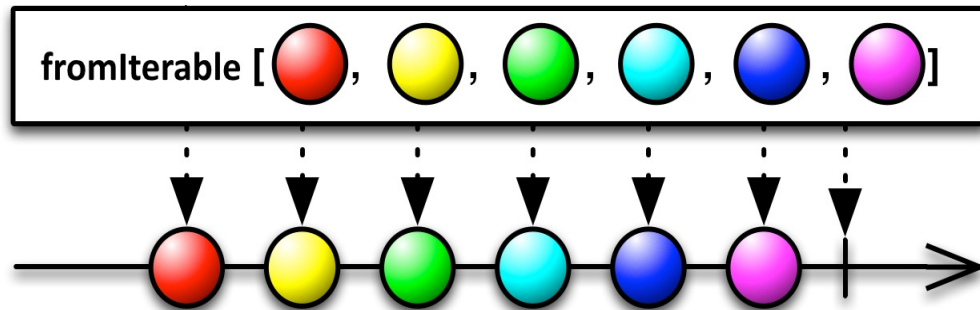
T - the type of elements returned by the iterator

All Known Subinterfaces:

BeanContext, BeanContextServices,
BlockingDeque<E>, BlockingQueue<E>,
Collection<E>, Deque<E>, DirectoryStream<T>,
List<E>, NavigableSet<E>, Path, Queue<E>,
SecureDirectoryStream<T>, Set<E>, SortedSet<E>,
TransferQueue<E>

Key Factory Method Operators in the Observable Class

- The `fromIterable()` operator
 - Create an Observable that emits the items contained in the given Iterable
- This factory method adapts non-reactive input sources into the reactive model
 - e.g., Java collections like List & Set



```
List<Integer> denominators =  
    List.of(3, 4, 2, 0 1);
```

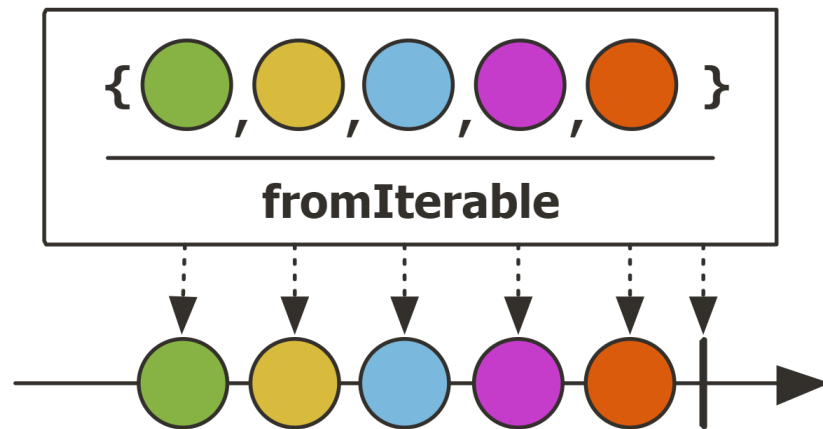
```
Observable  
    .fromIterable(denominators)  
    ...
```

Create an Observable stream of Integer objects from a Java List collection

See [Reactive/Observable/ex1/src/main/java/ObservableEx.java](#)

Key Factory Method Operators in the Observable Class

- The fromIterable() operator
 - Create an Observable that emits the items contained in the given Iterable
 - This factory method adapts non-reactive input sources into the reactive model
- Project Reactor's Flux.fromIterable() operator works the same



```
List<Integer> list =  
    List.of(0, 1, 1, 2, 3, 5, 8, 13, 21);
```

Create a Flux stream of Integer objects from a Java List collection

```
Flux  
    .fromIterable(list)  
    ...
```

Key Factory Method Operators in the Observable Class

- The fromIterable() operator
 - Create an Observable that emits the items contained in the given Iterable
 - This factory method adapts non-reactive input sources into the reactive model
 - Project Reactor's Flux.fromIterable() operator works the same
 - Similar to the Collection.stream() method in Java Streams

Create a stream of Integer objects

stream

```
default Stream<E> stream()
```

Returns a sequential Stream with this collection as its source.

This method should be overridden when the `spliterator()` method cannot return a spliterator that is IMMUTABLE, CONCURRENT, or *late-binding*. (See `spliterator()` for details.)

Implementation Requirements:

The default implementation creates a sequential Stream from the collection's Spliterator.

Returns:

a sequential Stream over the elements in this collection

```
List<Integer> list =  
    List.of(0,1,1,2,3,5,8,13,21);
```

```
list.stream() ...
```

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

Key Factory Method Operators in the Observable Class

- The fromCallable() operator
 - Returns an Observable that, when an observer subscribes to it, does certain things

```
static <T> Observable<T>  
    fromCallable(Callable<? extends T>  
                callable)
```

Key Factory Method Operators in the Observable Class

- The fromCallable() operator
 - Returns an Observable that, when an observer subscribes to it, does certain things
 - Invokes a Callable param

```
static <T> Observable<T>  
    fromCallable(Callable<? extends T>  
                callable)
```

Interface Callable<V>

Type Parameters:

V - the result type of method call

All Known Subinterfaces:

DocumentationTool.DocumentationTask,
JavaCompiler.CompilationTask

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/concurrent/Callable.html

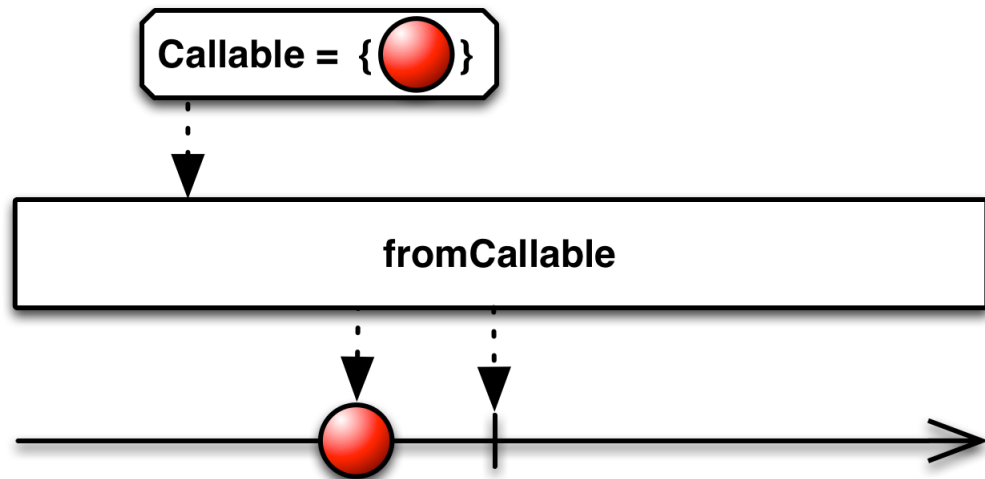
Key Factory Method Operators in the Observable Class

- The fromCallable() operator
 - Returns an Observable that, when an observer subscribes to it, does certain things
 - Invokes a Callable param
 - The returned Observable emits the value returned from the Callable

```
static <T> Observable<T>  
    fromCallable(Callable<? extends T>  
                callable)
```

Key Factory Method Operators in the Observable Class

- The fromCallable() operator
 - Returns an Observable that, when an observer subscribes to it, does certain things
- This factory method adapts non-reactive input sources into the reactive model



Observable

.fromCallable

((

```
-> BigFractionUtils  
    .makeBigFraction(random,  
                      true))
```

Create an Observable that emits one random BigFraction

See [Reactive/Observable/ex1/src/main/java/ObservableEx.java](https://github.com/reactive/reactive-examples/blob/master/src/main/java/Reactive/Observable/ex1/src/main/java/ObservableEx.java)

Key Factory Method Operators in the Observable Class

- The fromCallable() operator
 - Returns an Observable that, when an observer subscribes to it, does certain things
 - This factory method adapts non-reactive input sources into the reactive model
 - This operator defers executing the Callable until an observer subscribes to the Observable
 - i.e., it is “lazy”



```
Observable
    .fromCallable
      ( ()
        -> BigFractionUtils
            .makeBigFraction(random,
                              true) )
```

Key Factory Method Operators in the Observable Class

- The `fromCallable()` operator
 - Returns an Observable that, when an observer subscribes to it, does certain things
 - This factory method adapts non-reactive input sources into the reactive model
- This operator defers executing the Callable until an observer subscribes to the Observable
 - i.e., it is "lazy"



Conversely, `Observable.just()` is "eager"

Observable

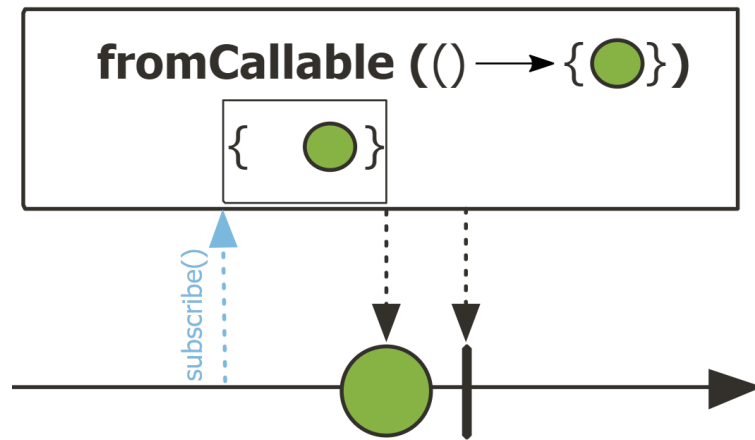
```
.just(BigFraction.valueOf(100,3),  
      BigFraction.valueOf(100,4),  
      BigFraction.valueOf(100,2),  
      BigFraction.valueOf(100,1))
```

...

Contrast with "eager" Observable factory method operators earlier in this lesson

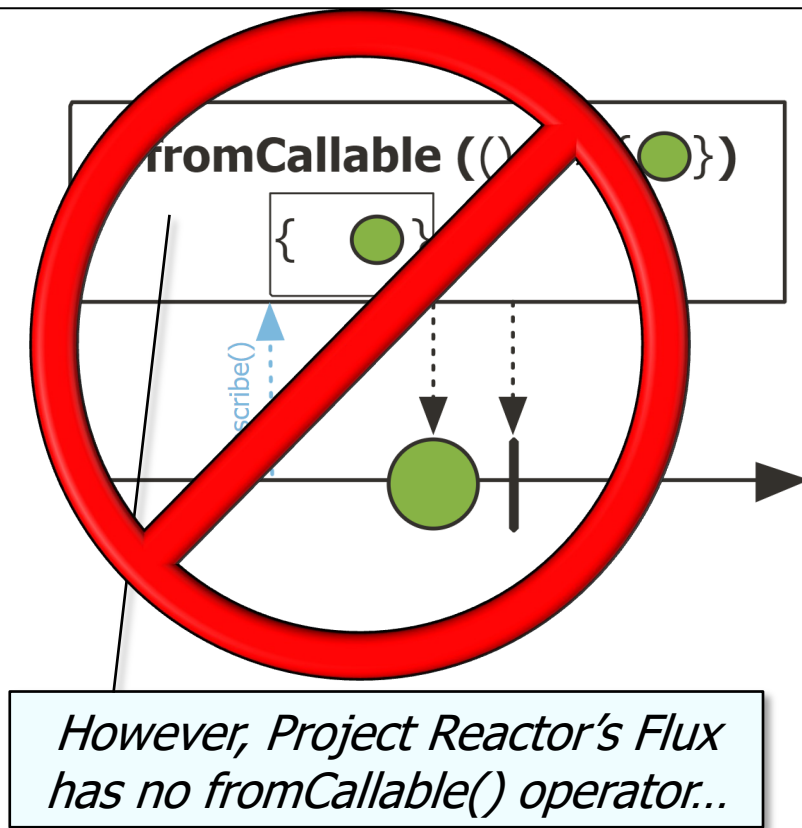
Key Factory Method Operators in the Observable Class

- The `fromCallable()` operator
 - Returns an Observable that, when an observer subscribes to it, does certain things
 - This factory method adapts non-reactive input sources into the reactive model
 - This operator defers executing the Callable until an observer subscribes to the Observable
 - Project Reactor's operator `Mono.fromCallable()` is similar



Key Factory Method Operators in the Observable Class

- The `fromCallable()` operator
 - Returns an Observable that, when an observer subscribes to it, does certain things
 - This factory method adapts non-reactive input sources into the reactive model
 - This operator defers executing the Callable until an observer subscribes to the Observable
- Project Reactor's operator `Mono.fromCallable()` is similar



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