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
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)

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Flowable.html#create
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()` method receives an `FlowableEmitter` instance

```java
public interface FlowableOnSubscribe<T> {
  void subscribe(@NonNull FlowableEmitter<T> emitter);
}
```

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**

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td><code>subscribe(@NonNull FlowableEmitter&lt;T&gt; emitter)</code></td>
</tr>
</tbody>
</table>

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)
```

See [reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/FlowableEmitter.html](reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/FlowableEmitter.html)
The `create()` operator

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

```java
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

```java
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 that emits elements from Flowable Emitter upon subscription

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

See medium.com/tompee/rxjava-ninja-hot-and-cold-observables-19b30d6cc2fa
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 that emits elements from `Flowable Emitter` upon subscription
    - Subject to the `BackpressureStrategy` mode

```java
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.

<table>
<thead>
<tr>
<th>Enum Constant Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enum Constant and Description</td>
</tr>
<tr>
<td>Buffer</td>
</tr>
<tr>
<td>Buffers all onNext values until the downstream consumes it.</td>
</tr>
<tr>
<td>Drop</td>
</tr>
<tr>
<td>Drops the most recent onNext value if the downstream can’t keep up.</td>
</tr>
<tr>
<td>Error</td>
</tr>
<tr>
<td>Signals a MissingBackpressureException in case the downstream can’t keep up.</td>
</tr>
<tr>
<td>Latest</td>
</tr>
<tr>
<td>Keeps only the latest onNext value, overwriting any previous value if the downstream can’t keep up.</td>
</tr>
<tr>
<td>Missing</td>
</tr>
<tr>
<td>The onNext events are written without any buffering or dropping.</td>
</tr>
</tbody>
</table>
```
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

```java
def 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

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

See <https://github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Flowable/ex1>
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

```java
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

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 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](http://jstobigdata.com/java/backpressure-in-project-reactor)
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

```java
public 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

See [docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#generate](docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#generate)
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
End of Key Factory Method Operators in the Flowable Class (Part 1)