Overview of the Overflow Strategies in the Project Reactor Flux Class

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

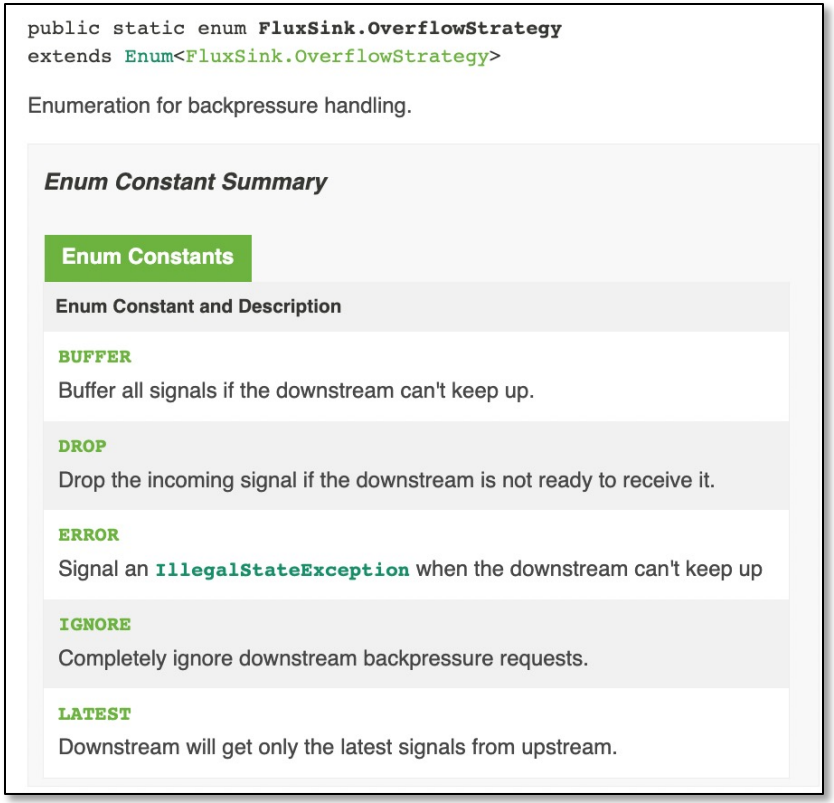
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

- Understand key classes in the Project Reactor API
- Know how Project Reactor Flux supports backpressure
- Recognize the Flux overflow strategies



```
public static enum FluxSink.OverflowStrategy
extends Enum<FluxSink.OverflowStrategy>
```

Enumeration for backpressure handling.

Enum Constant Summary

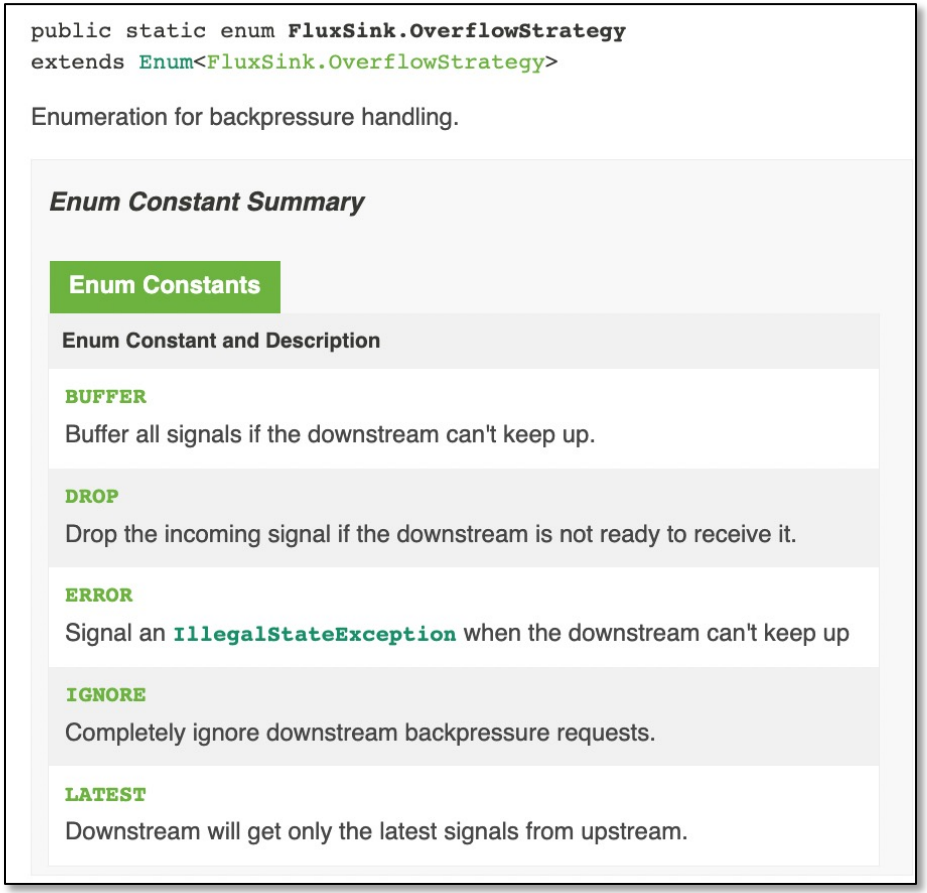
Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an `IllegalStateException` when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

Overview of Flux Over flow Strategies

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received



public static enum **FluxSink.OverflowStrategy**
extends Enum<FluxSink.OverflowStrategy>

Enumeration for backpressure handling.

Enum Constant Summary

Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an **IllegalStateException** when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

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

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received

All values are buffered so that subscriber can receive all values

```
public static enum FluxSink.OverflowStrategy
extends Enum<FluxSink.OverflowStrategy>
```

Enumeration for backpressure handling.

Enum Constant Summary

Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an `IllegalStateException` when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

May cause some delays in processing, but won't lose values (until memory is exhausted)

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received

Drop most recent onNext() value if down stream can't keep up because its too slow


```
public static enum FluxSink.OverflowStrategy
extends Enum<FluxSink.OverflowStrategy>
```

Enumeration for backpressure handling.

Enum Constant Summary

Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an `IllegalStateException` when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received

Throw OverflowException if the down stream can't keep up due to slowness (the documentation is incorrect here)

`public static enum FluxSink.OverflowStrategy extends Enum<FluxSink.OverflowStrategy>`

Enumeration for backpressure handling.

Enum Constant Summary

Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an `IllegalStateException` when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

See chat.openai.com/share/e347ec9d-ec85-47fc-9d40-b5b7928b1b7a

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received

There is no buffering or dropping, so Subscriber(s) must handle over flow or they will receive an error

```
public static enum FluxSink.OverflowStrategy
extends Enum<FluxSink.OverflowStrategy>
```

Enumeration for backpressure handling.

Enum Constant Summary

Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an `IllegalStateException` when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

See chat.openai.com/share/26d464f0-2f9a-46b6-8012-ea55bebc3113

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received

Only keep latest onNext() value, over writing previous value if downstream can't keep up because it's too slow

`public static enum FluxSink.OverflowStrategy extends Enum<FluxSink.OverflowStrategy>`

Enumeration for backpressure handling.

Enum Constant Summary

Enum Constants

Enum Constant and Description
BUFFER Buffer all signals if the downstream can't keep up.
DROP Drop the incoming signal if the downstream is not ready to receive it.
ERROR Signal an `IllegalStateException` when the downstream can't keep up
IGNORE Completely ignore downstream backpressure requests.
LATEST Downstream will get only the latest signals from upstream.

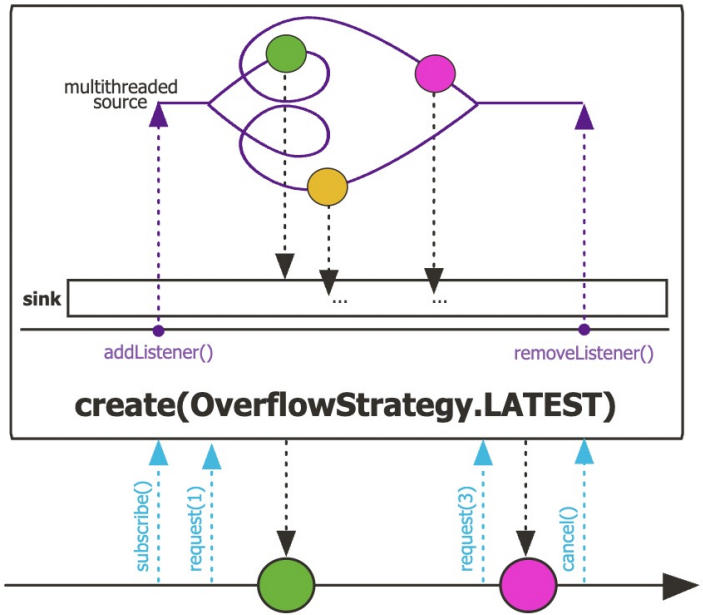
This strategy effectively has a "buffer" of size one..

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator

```
public static <T> Flux<T> create(Consumer<? super FluxSink<T>> emitter,
                                 FluxSink.OverflowStrategy backpressure)
```

Programmatically create a `Flux` with the capability of emitting multiple elements in a synchronous or asynchronous manner through the `FluxSink` API. This includes emitting elements from multiple threads.



This Flux factory is useful if one wants to adapt some other multi-valued async API and not worry about cancellation and backpressure (which is handled by buffering all signals if the downstream can't keep up).

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

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
    - Specify the overflow mode to apply if Subscriber can't keep up with Publisher

```
Flux
   .create(makeEmitter(count,
                               sb),
           FluxSink
           .OverflowStrategy
           .ERROR)

   .flatMap(bf1 ->
      multiplyFraction(bf1,
          sBigReducedFraction,
          Schedulers.parallel(),
          sb))

    .subscribe
      (blockingSubscriber);
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/flux/ex4

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
    - Specify the overflow mode to apply if Subscriber can't keep up with Publisher

```
Flux
   .create(makeEmitter(count,
                       sb),
           FluxSink
           .OverflowStrategy
           .ERROR)

   .flatMap(bf1 ->
      multiplyFraction(bf1,
          sBigReducedFraction,
          Schedulers.parallel(),
          sb))

   .subscribe
     (blockingSubscriber);
```

Rapidly emit a stream of random BigFraction objects in one fell swoop

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
    - Specify the overflow mode to apply if Subscriber can't keep up with Publisher

Throw exception when events can't be processed immediately

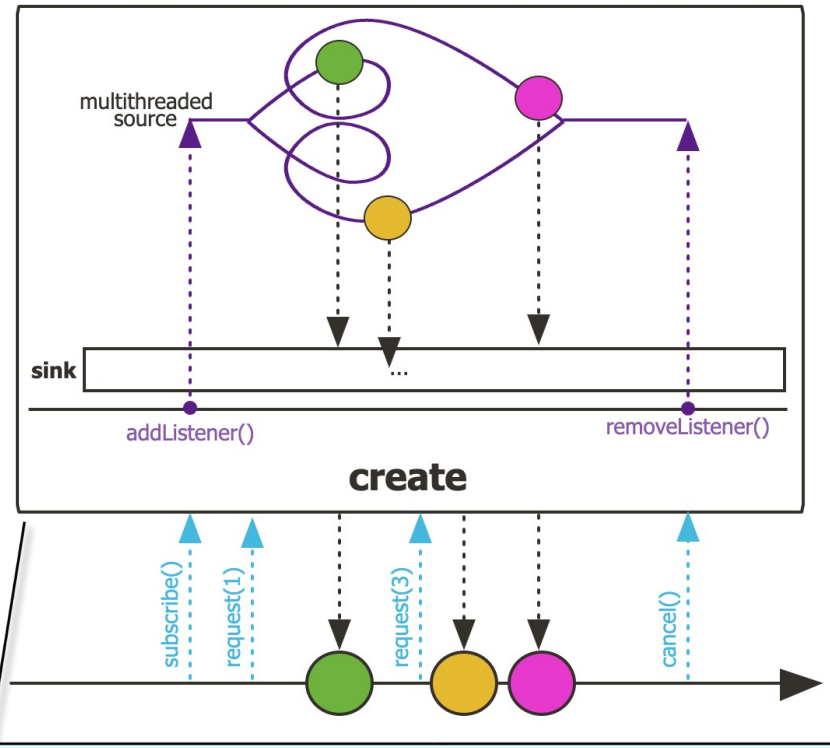
```
Flux
   .create(makeEmitter(count,
                               sb),
           FluxSink
           .OverflowStrategy
           .ERROR)

   .flatMap(bf1 ->
      multiplyFraction(bf1,
          sBigReducedFraction,
          Schedulers.parallel(),
          sb))

    .subscribe
      (blockingSubscriber);
```

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
    - Specify the overflow mode to apply if Subscriber can't keep up with Publisher
  - This operator is different than the one param version of Flux.create()



This Flux.create() operator just buffers all signals & does not support other backpressure strategies

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

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
  - They can also be provided via other Flux onBackpressure*() operators

- I want to deal with backpressure "errors" (request max from upstream and apply the strategy when downstream does not produce enough request)…
  - by throwing a special IllegalStateException: Flux#onBackpressureError
  - by dropping excess values: Flux#onBackpressureDrop
    - …except the last one seen: Flux#onBackpressureLatest
  - by buffering excess values (bounded or unbounded): Flux#onBackpressureBuffer
    - …and applying a strategy when bounded buffer also overflows: Flux#onBackpressureBuffer with a BufferOverflowStrategy

See projectreactor.io/docs/core/release/reference/#which.errors

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
  - They can also be provided via other Flux onBackpressure*() operators
    - onBackpressureDrop()
      - Ignore all streamed items that can't be processed until down stream can accept more of them

```
component
   .mouseMoves()
   .onBackpressureDrop()
   .publishOn
      (Schedulers.parallel(),
       1)
   .subscribe(event ->
                compute(event.x,
                        event.y));
```

See Flux.html#onBackpressureDrop

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
  - They can also be provided via other Flux onBackpressure*() operators
    - onBackpressureLatest()
      - Like the DROP strategy, but it keeps the last emitted item

```
component
   .mouseClicks()
   .onBackpressureLatest()
   .publishOn
      (Schedulers.parallel())
   .subscribe(event ->
                compute(event.x,
                        event.y),
                Throwable::
                printStackTrace);
```

See Flux.html#onBackpressureLatest

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
  - They can also be provided via other Flux onBackpressure*() operators
    - onBackpressureBuffer()
      - Creates a buffer to hold emitted items that can't be processed by downstream

```
Flux
   .range(1, 1_000_000)
   .onBackpressureBuffer
      (16,
       BufferOverflowStrategy
       .DROP_OLDEST)
   .publishOn
      (Schedulers.parallel())
   .subscribe(e -> { },
              Throwable::
              printStackTrace);
```

See Flux.html#onBackpressureBuffer

Overview of Flux Overflow Strategies

- Flux overflow strategies say how to handle emitted items that can't be processed as fast as they're received
  - These strategies can be provided via the two param version of the Flux.create() operator
  - They can also be provided via other Flux onBackpressure*() operators
    - onBackpressureBuffer()
      - Creates a buffer to hold emitted items that can't be processed by downstream
        - Buffer can be bounded or unbounded

```
Flux
   .range(1, 1_000_000)
   .onBackpressureBuffer
      (16,
       BufferOverflowStrategy
       .DROP_OLDEST)
   .publishOn
      (Schedulers.parallel())
   .subscribe(e -> { },
              Throwable::
              printStackTrace);
```

When buffer is full, remove oldest element from it & offer new element at end instead

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

End of Overview of Overflow Strategies in the Project Reactor Flux Class