The Image Counter Case Study (Part 1)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

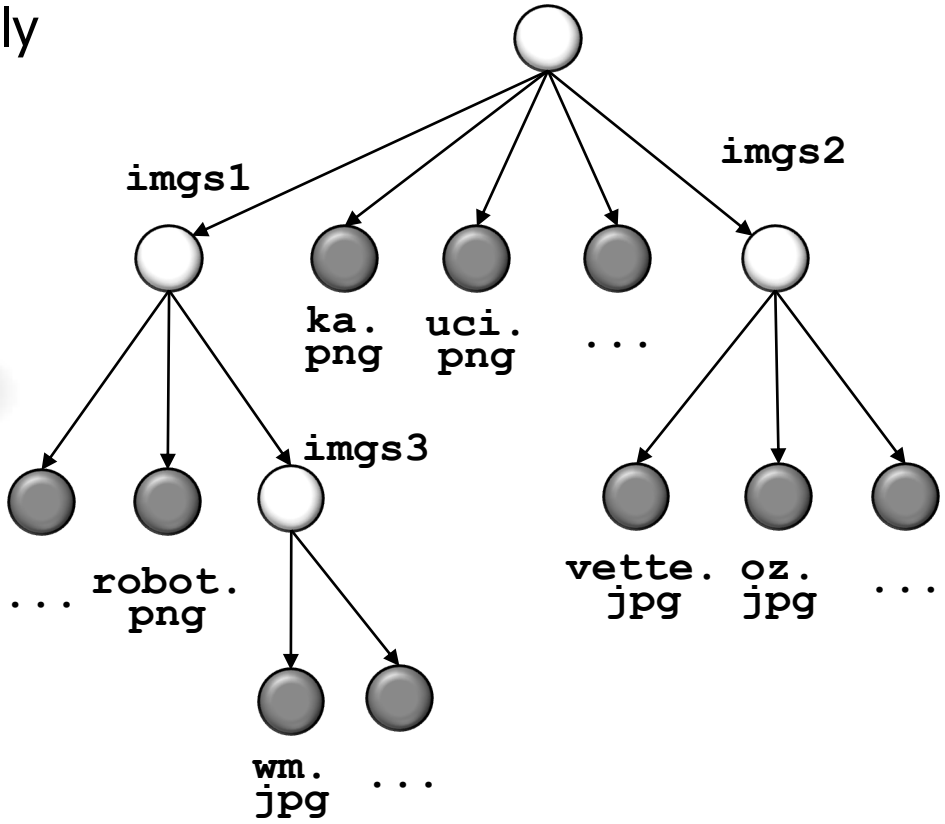
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

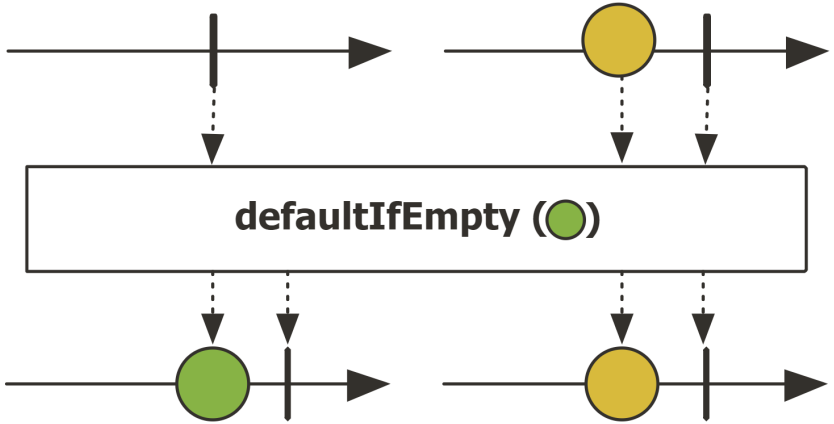
- Understand the structure & functionality of a program that crawls a hierarchical folder structure recursively

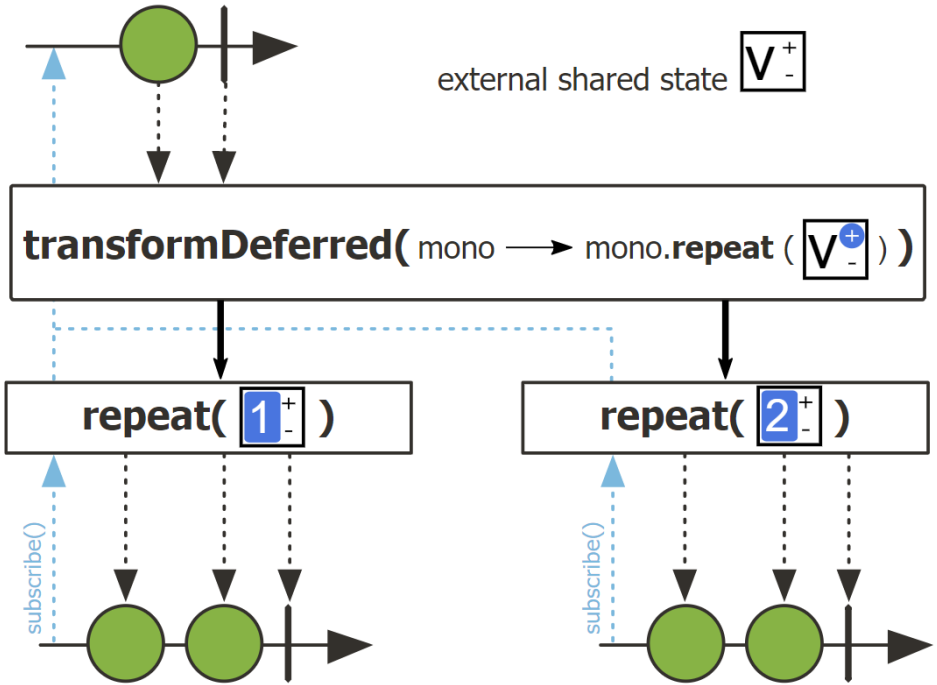


See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/ImageCounter

Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of a program that crawls a hierarchical folder structure recursively
  - Recognize other Project Reactor operators that are applied in the case study

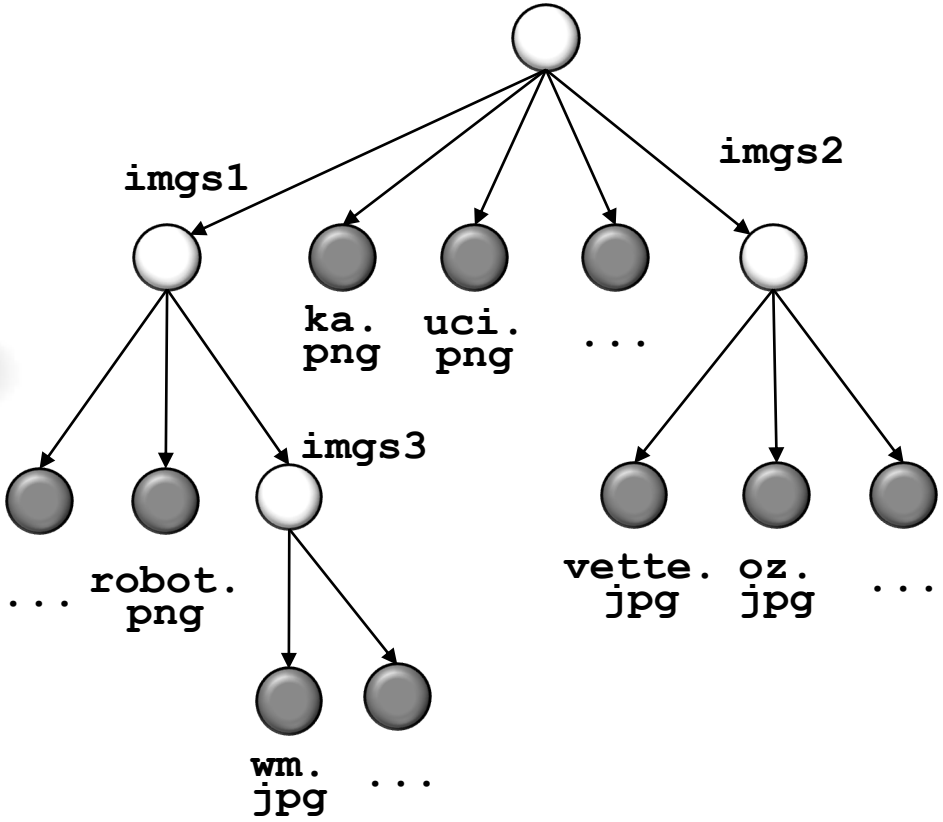




Applying Project Reactor Operators to the Image Crawler

Applying Project Reactor Operators to the Image Crawler

- This case study crawls a hierarchy of folders recursively

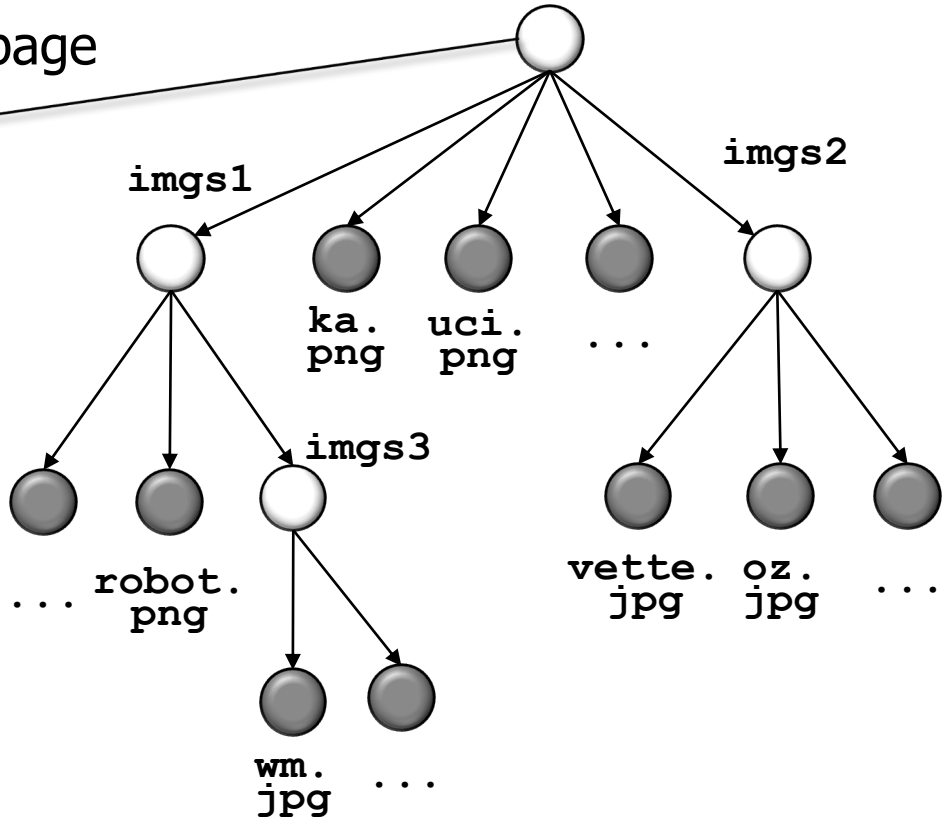


See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/ImageCounter

Applying Project Reactor Operators to the Image Crawler

- This case study crawls a hierarchy of folders recursively
  - It counts the # of images on each page

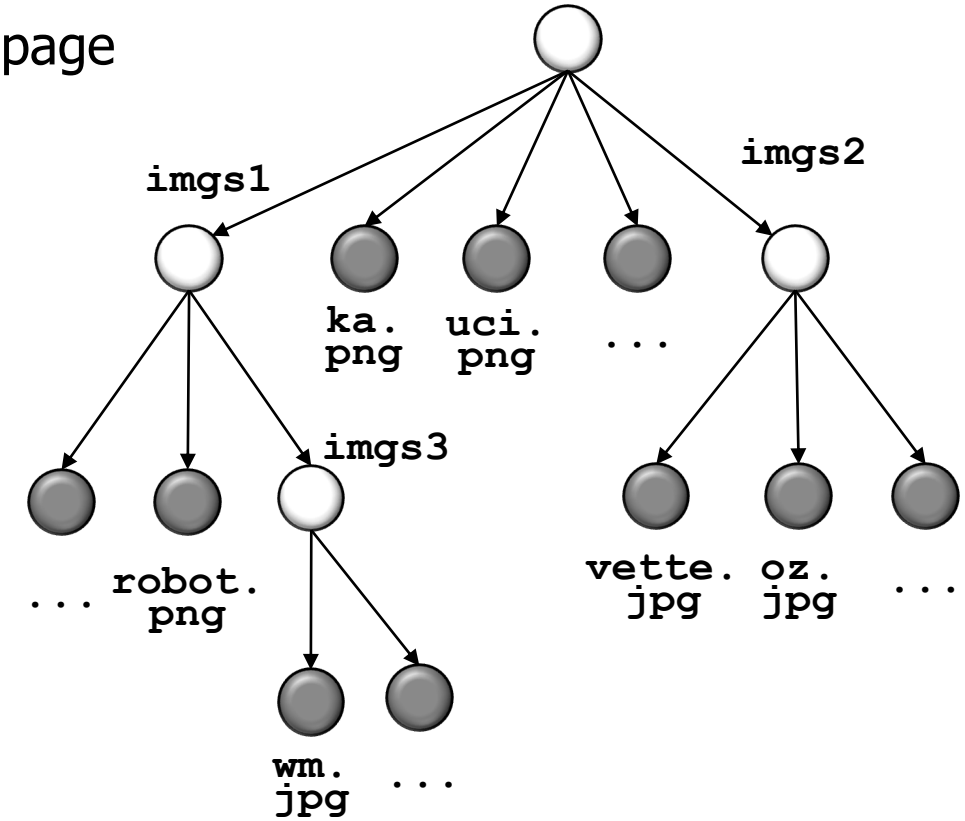
The root folder can either reside locally (filesystem-based) or be accessed remotely (web-based)



Applying Project Reactor Operators to the Image Crawler

- This case study crawls a hierarchy of folders recursively
  - It counts the # of images on each page

```
ImageCounter[Depth2]: Already
  processed imgs1/index.html
ImageCounter[Depth3]: Exceeded max
  depth of 2
ImageCounter[Depth2]: found 7 images
  for imgs1/index.html in thread 12
ImageCounter[Depth2]: found 7 images
  for imgs2/index.html in thread 13
ImageCounter[Depth1]: found 21 images
  for index.html in thread 13
ImageCounter: 21 total image(s) are
  reachable from index.html
```



Applying Project Reactor Operators to the Image Counter

Applying Project Reactor Operators to the Image Counter

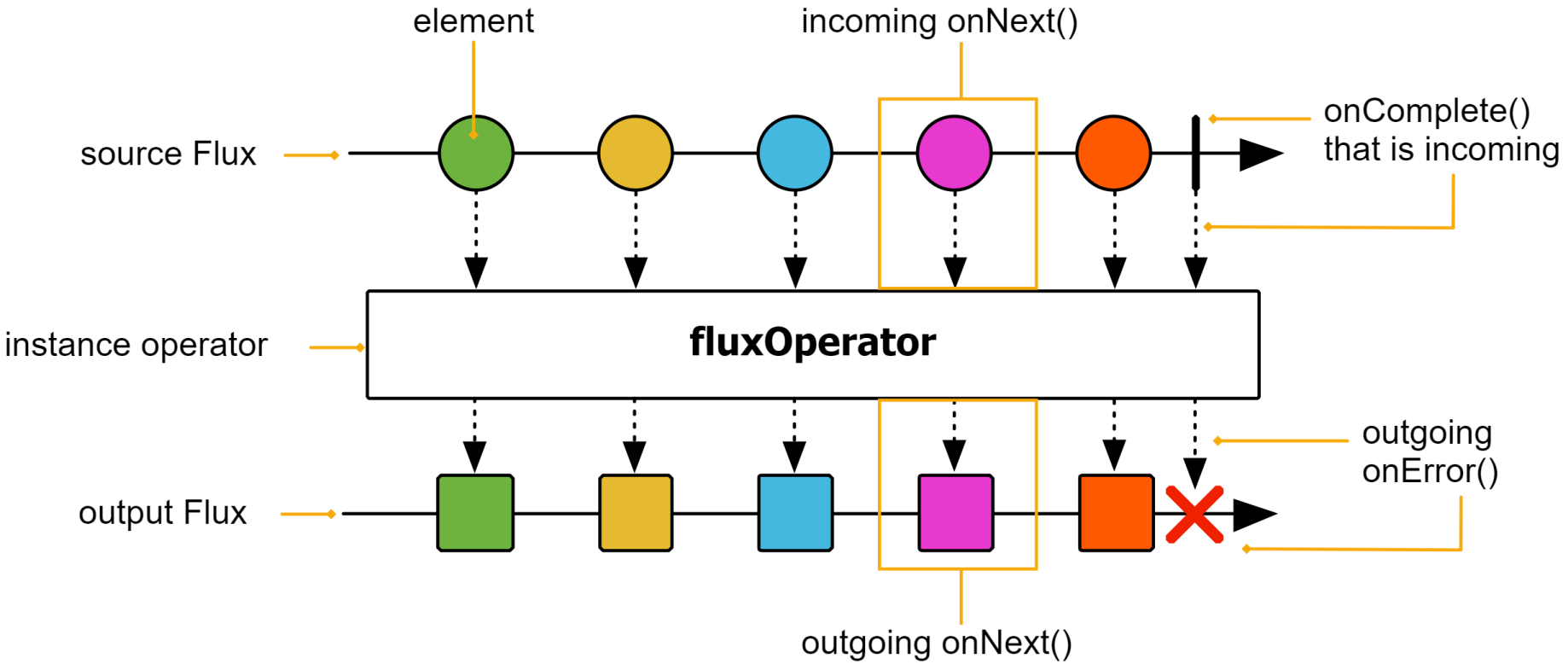
- The ImageCounter class defines methods for performing recursive traversal

Modifier and Type	Method	Description
private reactor.core.publisher.Mono<java.lang.Integer>	combineImageCounts (reactor.core.publisher.Mono<java.lang.Integer> imagesInPageMono, reactor.core.publisher.Mono<java.lang.Integer> imagesInLinksMono)	Count of the # of images on this page plus the # of images on hyperlinks accessible via this page.
private reactor.core.publisher.Mono<java.lang.Integer>	countImages(java.lang.String pageUri, int depth)	Main entry point into the logic for counting images asynchronously.
private reactor.core.publisher.Mono<java.lang.Integer>	countImagesAsync(java.lang.String pageUri, int depth)	Helper method that performs image counting asynchronously.
private reactor.core.publisher.Mono<java.lang.Integer>	crawlLinksInPage(org.jsoup.nodes.Document page, int depth)	Recursively crawl through hyperlinks that are in a page.
private org.jsoup.select.Elements	getImagesInPage(org.jsoup.nodes.Document page)	Factory method that returns a collection of IMG SRC URLs in this page
private reactor.core.publisher.Mono<org.jsoup.nodes.Document>	getStartPage(java.lang.String pageUri)	Factory method returns a mono to the page at the root URI

See Reactive/ImageCounter/src/main/java/ImageCounter.java

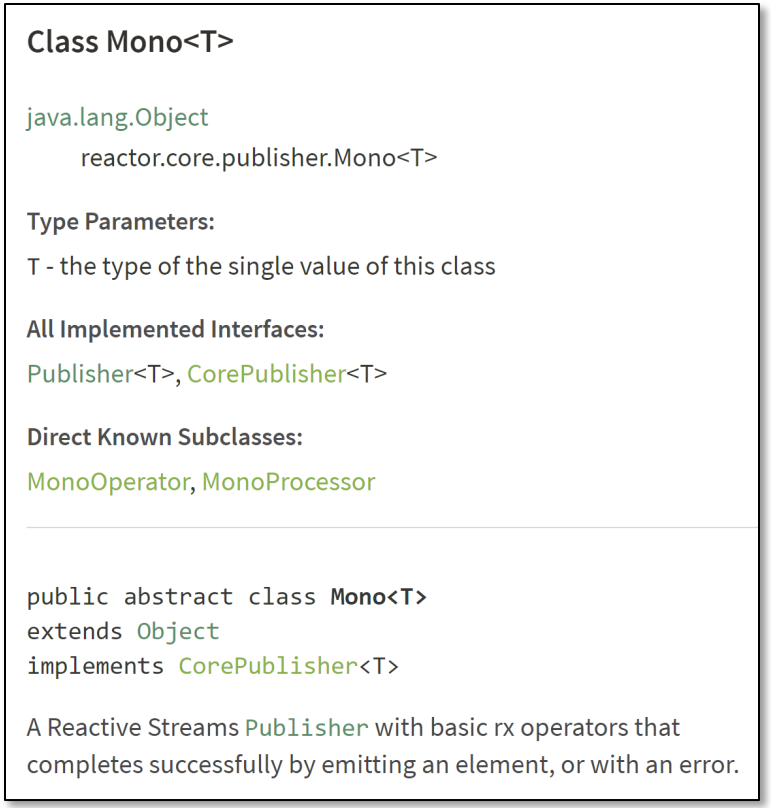
Applying Project Reactor Operators to the Image Counter

- Methods in the ImageCounter class use many Project Reactor operators



Applying Project Reactor Operators to the Image Counter

- Methods in the ImageCounter class use many Project Reactor operators
  - Mono operators
    - e.g., just(), block(), zipWith(), flatMap(), doOnSuccess(), map(), subscribeOn(), defaultIfEmpty(), transformDeferred(), blockOptional()



Class Mono<T>

java.lang.Object
    reactor.core.publisher.Mono<T>

Type Parameters:

T - the type of the single value of this class

All Implemented Interfaces:

Publisher<T>, CorePublisher<T>

Direct Known Subclasses:

MonoOperator, MonoProcessor

```
public abstract class Mono<T>
extends Object
implements CorePublisher<T>
```

A Reactive Streams `Publisher` with basic rx operators that completes successfully by emitting an element, or with an error.

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

Applying Project Reactor Operators to the Image Counter

- Methods in the ImageCounter class use many Project Reactor operators
  - Mono operators
  - Flux operators
    - e.g., fromIterable() & parallel()

Class Flux<T>

java.lang.Object
    reactor.core.publisher.Flux<T>

Type Parameters:

T - the element type of this Reactive Streams `Publisher`

All Implemented Interfaces:

Publisher<T>, CorePublisher<T>

Direct Known Subclasses:

ConnectableFlux, FluxOperator, FluxProcessor, GroupedFlux

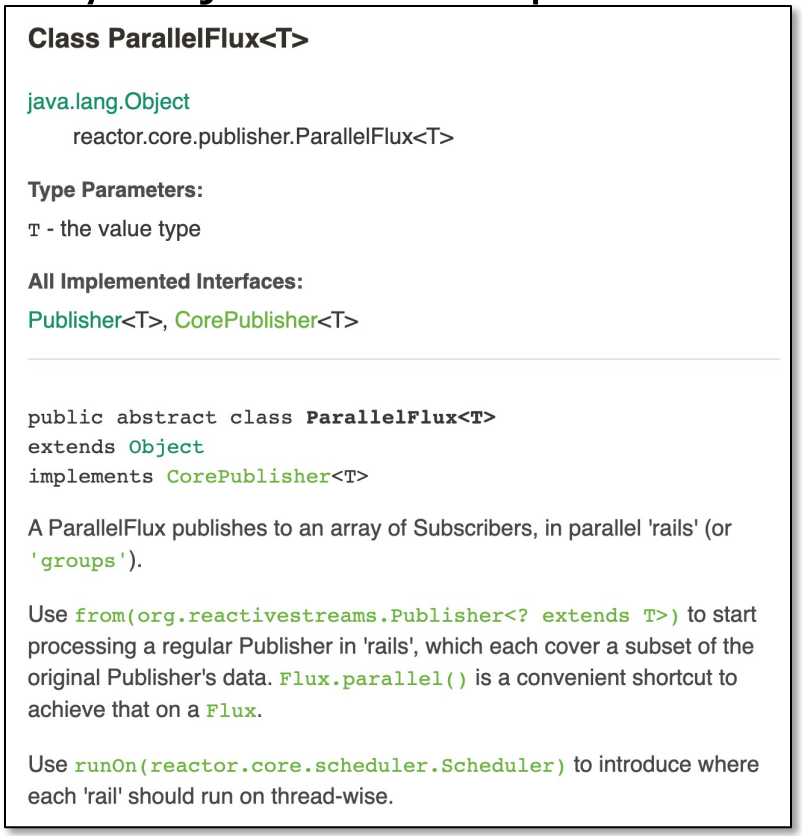
```
public abstract class Flux<T>
extends Object
implements CorePublisher<T>
```

A Reactive Streams `Publisher` with rx operators that emits 0 to N elements, and then completes (successfully or with an error).

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

Applying Project Reactor Operators to the Image Counter

- Methods in the ImageCounter class use many Project Reactor operators
  - Mono operators
  - Flux operators
  - ParallelFlux operators
    - e.g., flatMap(), runOn(), & reduce()



Class ParallelFlux<T>

java.lang.Object
reactor.core.publisher.ParallelFlux<T>

Type Parameters:

`T` - the value type

All Implemented Interfaces:

Publisher<T>, CorePublisher<T>

```
public abstract class ParallelFlux<T>
extends Object
implements CorePublisher<T>
```

A ParallelFlux publishes to an array of Subscribers, in parallel 'rails' (or `'groups'`).

Use `from(org.reactivestreams.Publisher<? extends T>)` to start processing a regular Publisher in 'rails', which each cover a subset of the original Publisher's data. `Flux.parallel()` is a convenient shortcut to achieve that on a `Flux`.

Use `runOn(reactor.core.scheduler.Scheduler)` to introduce where each 'rail' should run on thread-wise.

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

Applying Project Reactor Operators to the Image Counter

- The defaultIfEmpty() operator
  - Return a given default value if this Mono is completed without any data

```
Mono<T> defaultIfEmpty
   (T defaultV)
```

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#defaultIfEmpty

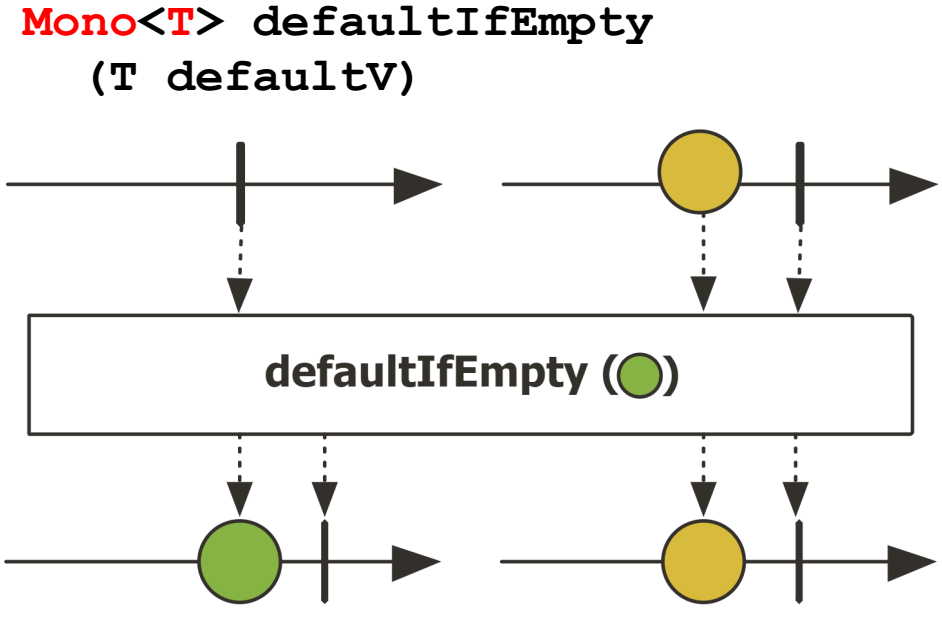
Applying Project Reactor Operators to the Image Counter

- The defaultIfEmpty() operator
  - Return a given default value if this Mono is completed without any data
    - The default value is passed as a param

```
Mono<T> defaultIfEmpty
   (T defaultV)
```

Applying Project Reactor Operators to the Image Counter

- The defaultIfEmpty() operator
  - Return a given default value if this Mono is completed without any data
    - The default value is passed as a param
  - Returns a new Mono that is either the value emitted or the default if no value was emitted



Applying Project Reactor Operators to the Image Counter

- The defaultIfEmpty() operator
  - Return a given default value if this Mono is completed without any data
  - Often used after combining operators that return empty results

```
return Flux
   .fromIterable(page
                 .select("a[href]"))

   .flatMap(hyperLink -> Mono
      .just(hyperLink)
      .transformDeferred(ReactorUtils
                         .commonPoolMono())
      .flatMap(url -> ...)

   .reduce(Integer::sum)

   .defaultIfEmpty(0);
```

Return 0 if there are no embedded images on an HTML page

See Reactive/ImageCounter/src/main/java/ImageCounter.java

Applying Project Reactor Operators to the Image Counter

- The defaultIfEmpty() operator
  - Return a given default value if this Mono is completed without any data
  - Often used after combining operators that return empty results
  - Similar to the Java Optional orElse() method

orElse

```
public T orElse(T other)
```

Return the value if present, otherwise return other.

Parameters:

other - the value to be returned if there is no value present, may be null

Returns:

the value, if present, otherwise other

See docs.oracle.com/javase/8/docs/api/java/util/Optional.html#orElse

Applying Project Reactor Operators to the Image Counter

- The Schedulers.fromExecutor() operator
  - Create a Scheduler that uses a backing Executor to schedule Runnables for async operators

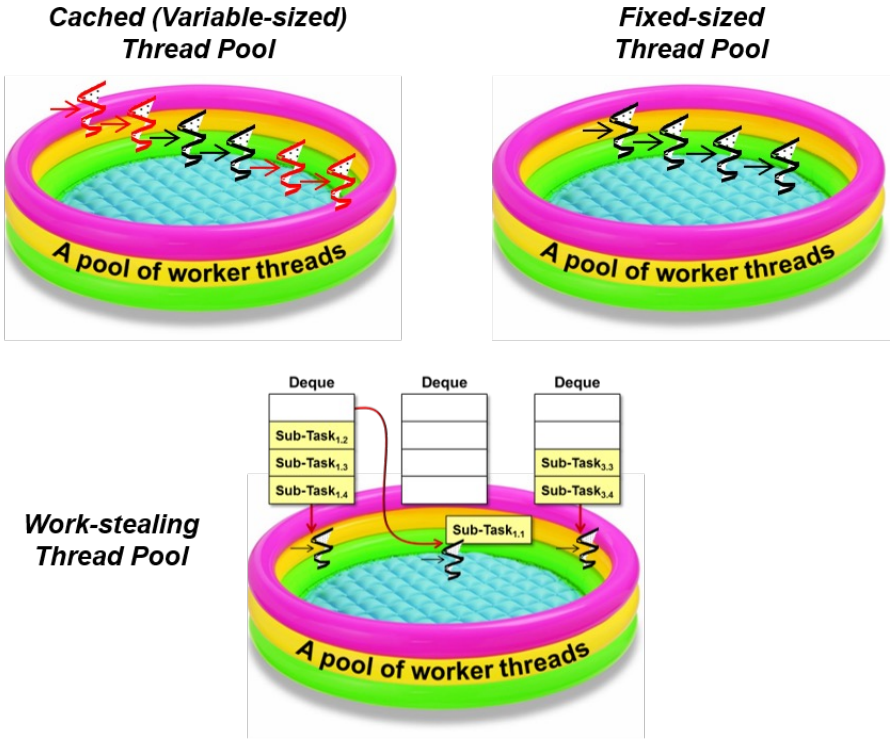
```
static Scheduler fromExecutor
  (Executor executor)
```

See projectreactor.io/docs/core/release/api/reactor/core/scheduler/Schedulers.html#fromExecutor

Applying Project Reactor Operators to the Image Counter

- The Schedulers.fromExecutor() operator
  - Create a Scheduler that uses a backing Executor to schedule Runnables for async operators
    - The Executor param can be the common fork-join pool or any other Executor implementation

```
static Scheduler fromExecutor
(Executor executor)
```



See docs.oracle.com/javase/tutorial/essential/concurrency/pools.html

Applying Project Reactor Operators to the Image Counter

- The Schedulers.fromExecutor() operator
  - Create a Scheduler that uses a backing Executor to schedule Runnables for async operators
    - The Executor param can be the common fork-join pool or any other Executor implementation

```
class ReactorUtils {
  public static <T> Function
                      <Mono<T>,
                       Mono<T>>
  commonPoolMono() {
    return mono -> mono
      .subscribeOn(Schedulers
                   .fromExecutor
                     (ForkJoinPool
                      .commonPool()));
  }

  ...
}
```

Schedule a Mono to run on the common fork-join pool

See Reactive/ImageCounter/src/main/java/utils/ReactorUtils.java

Applying Project Reactor Operators to the Image Counter

- The Schedulers.fromExecutor() operator
  - Create a Scheduler that uses a backing Executor to schedule Runnables for async operators
    - The Executor param can be the common fork-join pool or any other Executor implementation

```
class ReactorUtils {
  public static <T> Function
                          <Mono<T>,
                           Mono<T>>
  commonPoolMono() {
    return mono -> mono
      .subscribeOn(Schedulers
                   .fromExecutor
                     (ForkJoinPool
                      .commonPool()));
  }

  ...
}
```

Returns the one-and-only common pool instance

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/ForkJoinPool.html#commonPool

Applying Project Reactor Operators to the Image Counter

- The transformDeferred() operator
  - Defer transformation of this Mono to generate a target Mono type

```
<V> Mono<V> transformDeferred
  (Function<? super Mono<T>,
            ? extends
              Publisher<V>>
    transformer)
```

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html#transformDeferred

Applying Project Reactor Operators to the Image Counter

- The transformDeferred() operator
  - Defer transformation of this Mono to generate a target Mono type
    - The Function param lazily maps this Mono into a target Publisher instance

```
<V> Mono<V> transformDeferred
  (Function<? super Mono<T>,
            ? extends
              Publisher<V>>
    transformer)
```

Interface Function<T,R>

Type Parameters:

T - the type of the input to the function

R - the type of the result of the function

All Known Subinterfaces:

UnaryOperator<T>

Functional Interface:

This is a functional interface and can therefore used as the assignment target for a lambda expression or method reference.

See docs.oracle.com/javase/8/docs/api/java/util/function/Function.html

Applying Project Reactor Operators to the Image Counter

- The transformDeferred() operator
  - Defer transformation of this Mono to generate a target Mono type
  - Can define "custom" operators used in a Mono pipeline

```
var imagesInLinksMono = pageMono
    .flatMap(page ->
       crawlLinksInPage(page))

  .transformDeferred
     (ReactorUtils
      .commonPoolMono());
```

Crawl links a web page via the common fork-Join pool

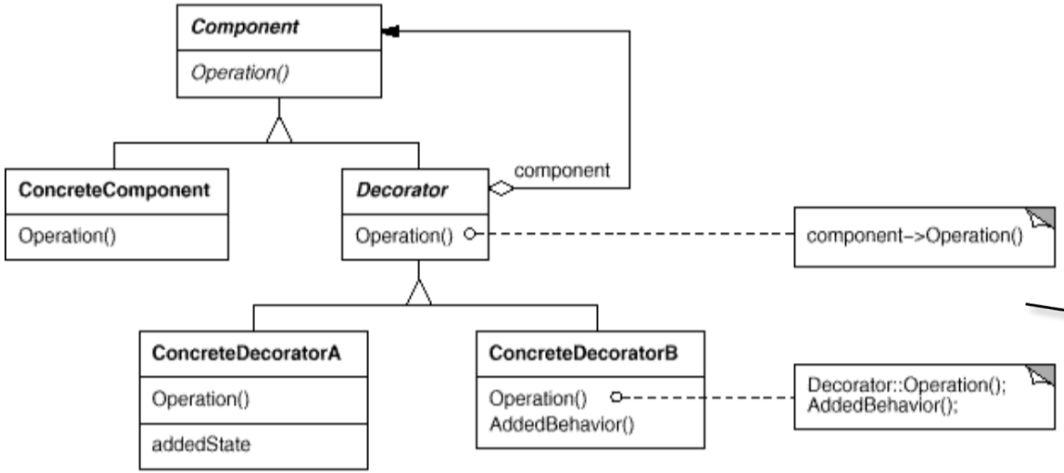
See Reactive/ImageCounter/src/main/java/ImageCounter.java

Applying Project Reactor Operators to the Image Counter

- The transformDeferred() operator
  - Defer transformation of this Mono to generate a target Mono type
  - Can define "custom" operators used in a Mono pipeline

```
var imagesInLinksMono = pageMono
    .flatMap(page ->
        crawlLinksInPage(page))

    .transformDeferred
        (ReactorUtils
         .commonPoolMono());
```

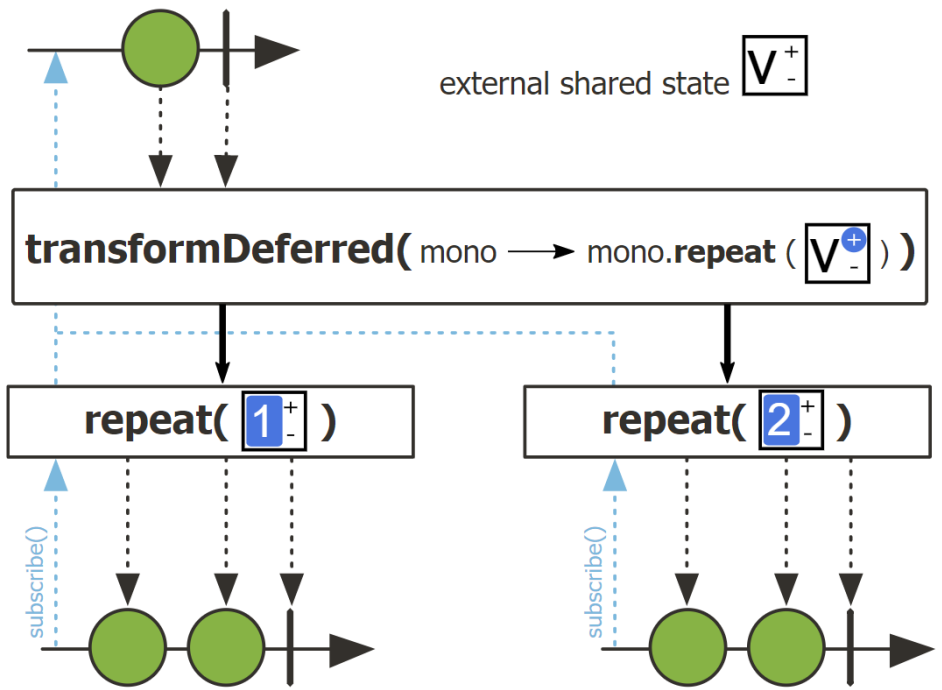


Implements the Decorator pattern that adds behavior to an object dynamically

See en.wikipedia.org/wiki/Decorator_pattern

Applying Project Reactor Operators to the Image Counter

- The transformDeferred() operator
  - Defer transformation of this Mono to generate a target Mono type
  - Can define "custom" operators used in a Mono pipeline
  - A transformation will occur for each Subscriber



Applying Project Reactor Operators to the Image Counter

- The transformDeferred() operator
  - Defer transformation of this Mono to generate a target Mono type
  - Can define "custom" operators used in a Mono pipeline
  - A transformation will occur for each Subscriber
    - The operator can produce a different operator chain for each subscription
      - e.g., by maintaining state or using a conditional expression

```
boolean para = ...

var imagesInLinksMono = pageMono
   .flatMap(page ->
      crawlLinksInPage(page))

  .transformDeferred
     (ReactorUtils
      .commonPoolMonoIf(para));
```

Use concurrent processing if para is true, else use sequential processing

See Reactive/ImageCounter/src/main/java/utils/ReactorUtils.java

Applying Project Reactor Operators to the Image Counter

- The transformDeferred() operator
  - Defer transformation of this Mono to generate a target Mono type
  - Can define "custom" operators used in a Mono pipeline
  - A transformation will occur for each Subscriber
    - The operator can produce a different operator chain for each subscription
      - e.g., by maintaining state or using a conditional expression

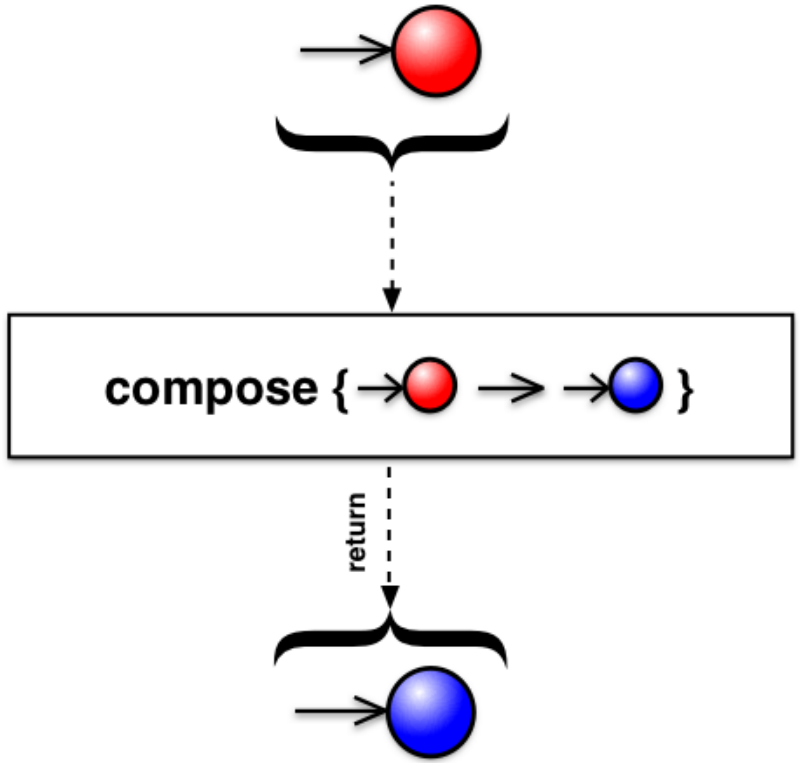
```
static <T> Function<Mono<T>,
                    Mono<T>>
commonPoolMonoIf(boolean para) {
  if (para)
    return ReactorUtils
      .commonPoolMono();
  else
    return mono -> mono;
}
```

Use concurrent processing if para is true, else use sequential processing

See Reactive/ImageCounter/src/main/java/utils/ReactorUtils.java

Applying Project Reactor Operators to the Image Counter

- The transformDeferred() operator
  - Defer transformation of this Mono to generate a target Mono type
  - Can define "custom" operators used in a Mono pipeline
  - A transformation will occur for each Subscriber
  - RxJava's Single.compose() operator works the same way



See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Single.html#compose

End of the Image Counter Case Study (Part 1)