Understanding Key Classes in the Project Reactor API

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

**Mono**

This is the timeline of the Mono
Time flows from left to right

This is the optional item emitted by the Mono

This vertical line indicates that the Mono has completed successfully

**Flux**

These are the items emitted by the Flux

Time flows from left to right

This vertical line indicates that the Flux has completed successfully

These dotted lines and this box indicate that a transformation is being applied to the Flux

The text inside the box shows the nature of the transformation

If for some reason the Flux terminates abnormally, with an error, the vertical line is replaced by an X

**operator (...)**

This Flux is the result of the transformation

If for some reason the Mono terminates abnormally, with an error, the vertical line is replaced by an X
Key Classes in the Project Reactor API
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value

---

**Class Mono<T>**

```java
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.

The recommended way to learn about the Mono API and discover new operators is through the reference documentation, rather than through this javadoc (as opposed to learning more about individual operators). See the "which operator do I need?" appendix.

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html)
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
    - Similar to a Java Completable Future or an async Optional<T>

BigFraction unreducedFraction = makeBigFraction(...);

Mono
  .fromCallable(() -> BigFraction
    .reduce(unreducedFraction))
  .subscribeOn
    (Schedulers.single())
  .map(result ->
    result.toMixedString())
  .doOnSuccess(result ->
    System.out.println
      ("big fraction = " + result + "\n"));

See stackoverflow.com/questions/54866391/makebigfraction
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional\(<T>\)
      - Can be documented via a “marble diagram”

See [projectreactor.io/docs/core/release/reference/#howtoReadMarbles](projectreactor.io/docs/core/release/reference/#howtoReadMarbles)
There are two key classes in the Project Reactor API:

- **Mono**
  - Completes successfully or with failure, may or may not emit a single value
    - Similar to a Java Completable Future or an async Optional<T>
    - Can be documented via a “marble diagram”
There are two key classes in the Project Reactor API

**Mono**
- Completes successfully or with failure, may or may not emit a single value
  - Similar to a Java Completable Future or an async Optional<T>
  - Can be documented via a “marble diagram”
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>
      - Can be documented via a “marble diagram”
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>
    - Can be documented via a “marble diagram”

This Mono is the result of the transformation
There are two key classes in the Project Reactor API:

- **Mono**
  - Completes successfully or with failure, may or may not emit a single value
    - Similar to a Java Completable Future or an async Optional<T>
    - Can be documented via a “marble diagram”

**Diagram:**

- Vertical line indicates the Mono completed successfully
- Operator (...)
- Error symbol
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
  - **Mono**
    - Completes successfully or with failure, may or may not emit a single value
      - Similar to a Java Completable Future or an async Optional<T>
      - Can be documented via a “marble diagram”

*If the Mono terminates abnormally the vertical line is replaced by an X*
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API

**Mono**
- Completes successfully or with failure, may or may not emit a single value
  - Similar to a Java Completable Future or an async Optional\(<T>\)
  - Can be documented via a “marble diagram”
- Provides a wide range of operators

- Factory method operators
- Transforming operators
- Action operators
- Concurrency & scheduler operators
- Combining operators
- Suppressing operators
- Blocking operators
- etc.
There are two key classes in the Project Reactor API:

- **Mono**
- **Flux**

  - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html)
There are two key classes in the Project Reactor API:

- **Mono**
- **Flux**

**Flux**
- Emits an indefinite number of events (0 to infinite) & may complete successfully or with failure.
- Similar to an async Java stream.
  - i.e., completable futures used with a Java stream.

```java
Flux
    .create
    (bigFractionEmitter)
    .take(sMAX_FRACTIONS)
    .flatMap(unreducedFraction ->
        reduceAndMultiplyFraction
        (unreducedFraction,
        Schedulers.parallel()))
    .collectList()
    .flatMap(list ->
        BigFractionUtils
        .sortAndPrintList
        (list, sb));
```
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
  - Mono
  - Flux
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
      - The subscriber indicates to the publisher how much data it can consume

See [jstobigdata.com/java/backpressure-in-project-reactor](jstobigdata.com/java/backpressure-in-project-reactor)
There are two key classes in the Project Reactor API

- **Mono**

- **Flux**
  - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
  - Similar to an async Java stream
  - Supports backpressure
  - Can also be documented via a marble diagram

See medium.com/@jshvarts/read-marble-diagrams-like-a-pro-3d72934d3ef5
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
  - Mono
  - Flux
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
    - Can also be documented via a marble diagram
There are two key classes in the Project Reactor API:

- **Mono**
- **Flux**

  - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
  - Similar to an async Java stream
  - Supports backpressure
  - Can also be documented via a marble diagram
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
  - Mono
  - Flux

- Flux
  - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
  - Similar to an async Java stream
  - Supports backpressure
  - Can also be documented via a marble diagram

These dotted lines & this box indicate that a transformation is being applied to the Flux

The text inside the box indicates the type of transformation
Key Classes in the Project Reactor API

• There are two key classes in the Project Reactor API
  • Mono
  • Flux
    • Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    • Similar to an async Java stream
    • Supports backpressure
    • Can also be documented via a marble diagram

These Flux elements are the result of the transformation
Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
  - Mono
  - Flux
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
    - Can also be documented via a marble diagram

This vertical line indicates the Flux completed successfully
There are two key classes in the Project Reactor API:

- **Mono**

- **Flux**
  - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
  - Similar to an async Java stream
  - Supports backpressure
  - Can also be documented via a marble diagram

*If the Flux terminates abnormally the vertical line is replaced by an X*
### Key Classes in the Project Reactor API

- There are two key classes in the Project Reactor API
  - **Mono**
  - **Flux**
    - Emits an indefinite # of events (0 to infinite) & may complete successfully or w/failure
    - Similar to an async Java stream
    - Supports backpressure
    - Can also be documented via a marble diagram
    - Provides a wide range of operators
  - Factory method operators
  - Transforming operators
  - Action operators
  - Concurrency & scheduler operators
  - Combining operators
  - Terminal operators
  - Suppressing operators
  - Blocking operators
  - etc.
End of Understanding Key Classes in the Project Reactor API