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

Flux
Key Classes in the Project Reactor API
There are two 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.

See [projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html](http://projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html)
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>

```java
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/mono-vs-completablefuture](https://stackoverflow.com/questions/54866391/mono-vs-completablefuture)
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 medium.com/@jshvarts/read-marble-diagrams-like-a-pro-3d72934d3ef5
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 is the timeline of a Mono, where time flows from left to right.
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”

This Mono is the result of the transformation
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 vertical line indicates the Mono completed successfully
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.
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

See 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 # of events (0 to infinite) & may complete successfully or w/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
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

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

This is the timeline of a Flux, where time flows from left to right
There are two key classes in the Project Reactor API:

- **Mono**
- **Flux**
  - Emits an indefinite number of events (0 to infinite) & may complete successfully or with 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 number of events (0 to infinite) & may complete successfully or with 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
  - 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
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

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