

# Difference Between Flux and Mono

Last updated: May 11, 2024



Written by: baeldung (<https://www.baeldung.com/author/baeldung>)



Reviewed by: Saajan Nagendra  
(<https://www.baeldung.com/editor/saajannagendra>)

**Reactive** (<https://www.baeldung.com/category/reactive>)

**Flux** (<https://www.baeldung.com/tag/flux>)

**Mono** (<https://www.baeldung.com/tag/mono>)

**Reactor** (<https://www.baeldung.com/tag/reactor>)



Spring 5 added support for reactive programming with the Spring WebFlux module, which has been improved upon ever since. Get started with the Reactor project basics and **reactive programming in Spring Boot**:

**>> Download the E-book** ([/eBook-Reactive-NPI-2rmn2](#))

# 1. Overview

In this tutorial, we'll learn the difference between *Flux*

(<https://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html>) and *Mono*

(<https://projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html>) of the Reactor Core (/reactor-core) library.

## 2. What Is *Mono*?

*Mono* is a special type of *Publisher* (<https://www.reactive-streams.org/reactive-streams-1.0.3-javadoc/org/reactivestreams/Publisher.html>).

**A *Mono* object represents a single or empty value.** This means it can emit only one value at most for the *onNext()* request and then terminates with the *onComplete()* signal. In case of failure, it only emits a single *onError()* signal.

Let's see an example of *Mono* with a completion signal:

```
@Test
public void givenMonoPublisher_whenSubscribeThenReturnSingleValue() {
    Mono<String> helloMono = Mono.just("Hello");
    StepVerifier.create(helloMono)
        .expectNext("Hello")
        .expectComplete()
        .verify();
}
```

We can see here that when *helloMono* is subscribed, it emits only one value and then sends the signal of completion.

## 3. What Is *Flux*?

*Flux* is a standard *Publisher* that represents 0 to N asynchronous sequence values. This means that **it can emit 0 to many values, possibly infinite values for *onNext()* requests, and then terminates with either a completion or an**

**error signal.**

Let's see an example of *Flux* with a completion signal:

```
@Test
public void givenFluxPublisher_whenSubscribedThenReturnMultipleValues() {
    Flux<String> stringFlux = Flux.just("Hello", "Baeldung");
    StepVerifier.create(stringFlux)
        .expectNext("Hello")
        .expectNext("Baeldung")
        .expectComplete()
        .verify();
}
```

Now, let's see an example of *Flux* with an error signal:

```
@Test
public void
givenFluxPublisher_whenSubscribeThenReturnMultipleValuesWithError() {
    Flux<String> stringFlux = Flux.just("Hello", "Baeldung", "Error")
        .map(str -> {
            if (str.equals("Error"))
                throw new RuntimeException("Throwing Error");
            return str;
        });
    StepVerifier.create(stringFlux)
        .expectNext("Hello")
        .expectNext("Baeldung")
        .expectError()
        .verify();
}
```

We can see here that after getting two values from the *Flux*, we get an error.

## 4. *Mono vs. Flux*

*Mono* and *Flux* are both implementations of the *Publisher* interface. In simple terms, we can say that when we're doing something like a computation or making a request to a database or an external service, and expecting a maximum of one result, then we should use *Mono*.

When we're expecting multiple results from our computation, database, or external service call, then we should use *Flux*.

*Mono* is more relatable to the *Optional* (/java-optional) class in Java since it contains 0 or 1 value, and *Flux* is more relatable to *List* (/java-arraylist) since it can have N number of values.

## 5. Conclusion

In this article, we've learned the difference between *Mono* and *Flux*.

The code backing this article is available on GitHub. Once you're **logged in as a Baeldung Pro Member (/members/)**, start learning and coding on the project.

## COURSES

ALL COURSES (/COURSES/ALL-COURSES)

BAELDUNG ALL ACCESS (/COURSES/ALL-ACCESS)

[BAELDUNG ALL TEAM ACCESS \(/COURSES/ALL-ACCESS-TEAM\)](/COURSES/ALL-ACCESS-TEAM)

[THE COURSES PLATFORM \(HTTPS://COURSES.BAELDUNG.COM\)](HTTPS://COURSES.BAELDUNG.COM)

## SERIES

[JAVA "BACK TO BASICS" TUTORIAL \(/JAVA-TUTORIAL\)](/JAVA-TUTORIAL)

[JACKSON JSON SERIES \(/JACKSON\)](/JACKSON)

[APACHE HTTPCLIENT SERIES \(/HTTPCLIENT-SERIES\)](/HTTPCLIENT-SERIES)

[REST WITH SPRING SERIES \(/REST-WITH-SPRING-SERIES\)](/REST-WITH-SPRING-SERIES)

[SPRING PERSISTENCE SERIES \(/PERSISTENCE-WITH-SPRING-SERIES\)](/PERSISTENCE-WITH-SPRING-SERIES)

[SECURITY WITH SPRING \(/SECURITY-SPRING\)](/SECURITY-SPRING)

[SPRING REACTIVE SERIES \(/SPRING-REACTIVE-SERIES\)](/SPRING-REACTIVE-SERIES)

## ABOUT

[ABOUT BAELDUNG \(/ABOUT\)](/ABOUT)

[THE FULL ARCHIVE \(/FULL\\_ARCHIVE\)](/FULL_ARCHIVE)

[EDITORS \(/EDITORS\)](/EDITORS)

[OUR PARTNERS \(/PARTNERS/\)](/PARTNERS/)

[PARTNER WITH BAELDUNG \(/PARTNERS/WORK-WITH-US\)](/PARTNERS/WORK-WITH-US)

[EBOOKS \(/LIBRARY/\)](/LIBRARY/)

[FAQ \(HTTPS://WWW.BAELDUNG.COM/LIBRARY/FAQ\)](HTTPS://WWW.BAELDUNG.COM/LIBRARY/FAQ)

[BAELDUNG PRO \(/MEMBERS/\)](/MEMBERS/)

[TERMS OF SERVICE \(/TERMS-OF-SERVICE\)](/TERMS-OF-SERVICE)

[PRIVACY POLICY \(/PRIVACY-POLICY\)](/PRIVACY-POLICY)

[COMPANY INFO \(/BAELDUNG-COMPANY-INFO\)](/BAELDUNG-COMPANY-INFO)

[CONTACT \(/CONTACT\)](/CONTACT)