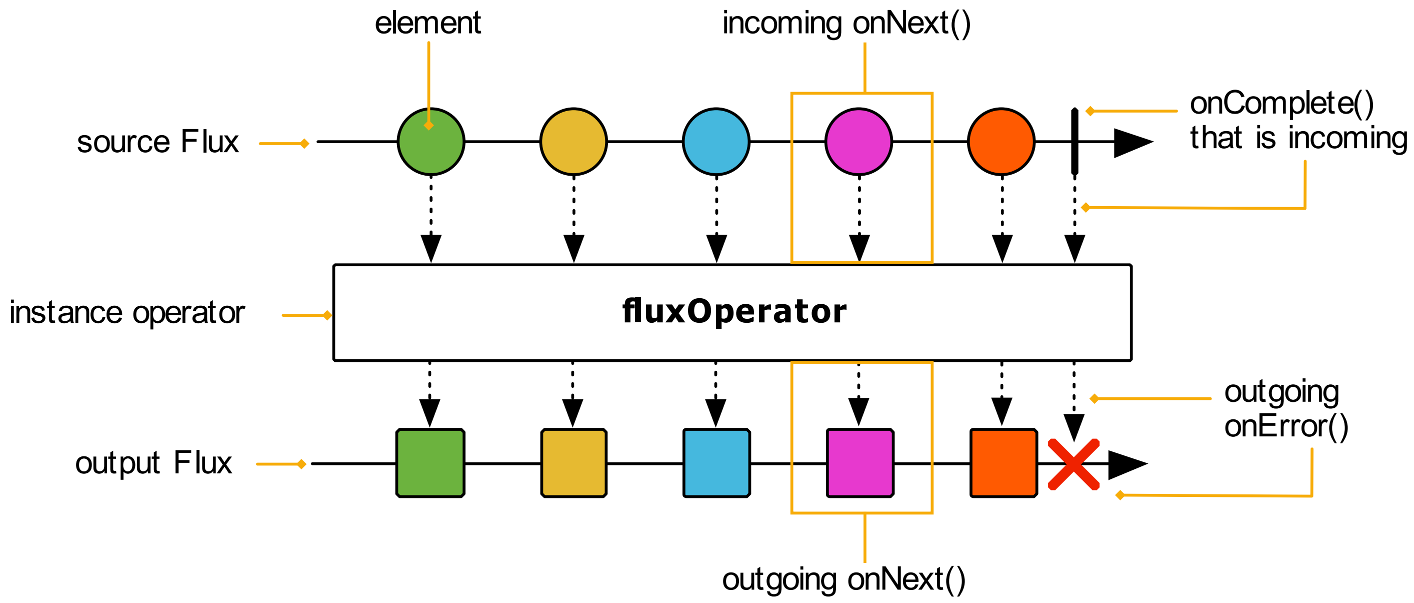
Reactive Programming in Java

Introduction:

The paradigm of programming which emphasizes the asynchronous flow of data is Reactive Programming. In Java, Project Reactor is one of the major frameworks that implements reactive programming. Reactive Programming is beneficial in scenarios where responsiveness, scalability and resilience are important.



Project Reactor:

This is one the most popular library for reactive Programming in Java and is also foundation of reactive programming in Spring Boot Ecosystem. It adheres to the Reactive Stream Specification and provides two main components.

Flux:

For Handling multiple items

Mono:

For Handling zero or one Item.

These components are the foundation for building reactive applications in Java.

e.g.

```java
package cc;

import reactor.core.publisher.Flux;

public class ReactiveBasics {

    public static void main(String[] args) {
        Flux<Integer> flux = Flux.just(1, 2, 3);
        flux.subscribe(System.out::println);
    }
}
```

This is the most basic example showing a Flux with three items. When subscribed to the flux, each of the items is passed in to println() function.

One of the main features of reactive programming is the ability to work with data in asynchronous and non-blocking manner, thereby allowing more efficient handling of events and responsiveness in application.

Reactive Programming usually involves the application of transformation and filtering operation on data streams. Project Reactor has a handful of operators that allows passing the data in the pipeline and applying various transformation in each of the different stages.

e.g.

```java
package cc;

import reactor.core.publisher.Flux;

public class TransformationAndFiltering {

    public static void main(String[] args) {
        Flux<Integer> flux = Flux.just(1, 2, 3, 4, 5);
        flux.map(i -> i * 2)
                .filter(i -> i > 5)
                .subscribe(System.out::println);
    }
}
```

Here, each item in flux is being passed on to next stage and transformation/filtering occurs in respective stages.

An interesting and most important concept of backpressure, which is essential for controlling the flow of data in scenarios where a producer is faster than the consumer, is also built in Project Reactor, thus ensuring the application can handle data at a pace suitable for the processing capacity.

```java
package cc;

import reactor.core.publisher.Flux;

public class BackpressureExample {

    public static void main(String[] args) {
        Flux.range(1, 1000)
                .doOnRequest(r -> System.out.println("Requesting: " + r))
                .limitRate(10)
                .subscribe(System.out::println);
    }
}
```

Here, the consumer specifies the rate limit of 10, so that the consumer is not overwhelmed with unbearable data from the producer.

Based on Reactive Specification, there are other libraries in java such as RxJava, Akka Stream. Each of these libraries has unique features and strengths, but the core concept is the same across all of these libraries.