Reactive Streams in Java

Before talking about how we can make use of this programming concept, let’s go through some basic concepts that are the base of understanding the scope of Reactive Streams

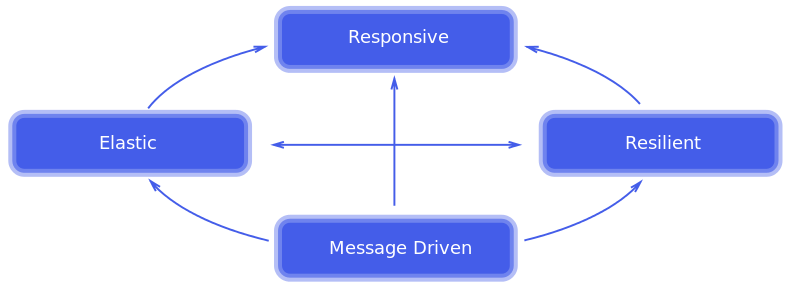
What is Reactive Programming?

*according to Wikipedia*

***In***[***computing***](https://en.wikipedia.org/wiki/Computing)***, reactive programming is a declarative***[***programming paradigm***](https://en.wikipedia.org/wiki/Programming_paradigm)***concerned with***[***data streams***](https://en.wikipedia.org/wiki/Dataflow_programming)***and the propagation of change. This means that it becomes possible to express static (e.g. arrays) or dynamic (e.g. event emitters) data streams with ease via the employed programming language(s), and that an inferred dependency within the associated execution model exists, which facilitates the automatic propagation of the change involved with data flow.***

***For example, in an imperative programming setting* A:=B + C*, {\displaystyle a:=b+c} would mean that A{\displaystyle a} is being assigned the result of*B + C *{\displaystyle b+c} in the instant the expression is evaluated, and later, the values of {\displaystyle b}*B *and/or*C *{\displaystyle c} can be changed with no effect on the value of*A *{\displaystyle a}. However, in reactive programming, the value of {\displaystyle a}*A *is automatically updated whenever the values of {\displaystyle b}*B *and/or*C *{\displaystyle c} change; without the program having to re-execute the sentence {\displaystyle a:=b+c} to determine the presently assigned value of*A *{\displaystyle a}.***

**Reactive Systems principals:**

****

In order to understand the principals behind the Reactive System please feel free to go through:

<https://www.reactivemanifesto.org/>

What are Reactive Streams?

The Reactive Streams is an initiative that defined a standardized API which offers to developers the tools they need to make use of Reactive Programming

Reactive Streams has become part of the JDK once with the release of Java 9 and consistent on the following functional Interfaces all grouped under the Flow class:

Subscriber Interface

public interface Subscriber<T> {

public void onSubscribe(Subscription s);

public void onNext(T t);

public void onError(Throwable t);

public void onComplete();

}

1. Invoked after a Publisher has completed the subscription for this Subscriber (but before sending any Subscription's item). The newly created Subscription object is passed via this method. The Subscriber typically assigns this instance to an instance variable for further use.
2. Invoked with a Subscription's next item of type T.
3. Invoked upon an unrecoverable error encountered by a Publisher or Subscription.
4. Invoked when no additional Subscriber method invocations will occur including onNext() method.

## Subscription Interface

This is the same interface whose instance is passed to the Subscriber.onSubscribe() method.

public interface Subscription {

public void request(long n);

public void cancel();

}

1. This is the key method behind non-blocking back-pressure concept. The Subscriber uses it to request n more items for consumption. This way the Subscriber controls how many items it is currently capable to receive (probably it will want to limit the consumption according to what resources it has).
2. Used by the Subscriber to cancel its subscription. After this call, no further items will be received.

## Publisher Interface

public interface Publisher<T> {

public void subscribe(Subscriber<? super T> s);

}

1. Used by the Subscribers for subscribing to receive the items of type T.

## Processor Interface

A component that acts as both a Subscriber and Publisher.

public interface Processor<T, R> extends Subscriber<T>, Publisher<R> {

}

## SubmissionPublisher class

This is the only concrete class provided in the Reactive Streams API. It implements the Publisher interface. We can use its submit() method to publish the provided item to each subscriber. In the following example we will see how to use this class and at the same time we will get familiar with the usage of above interfaces.

## Reactive Flow description

Bellow you can see a complete Flow:

* Create a **Publisher** and a **Subscriber**.
* Subscribe the subscriber with **Publisher::subscribe**.
* The publisher creates a Subscription and calls **Subscriber::onSubscription** with it so the subscriber can store the subscription.
* At some point the subscriber calls **Subscription::request** to request a number of items.
* The publisher starts handing items to the subscriber by calling **Subscriber::onNext**. It will never publish more than the requested number of items.
* The publisher might at some point be depleted or run into trouble and call **Subscriber::onComplete** or **Subscriber::onError**, respectively.
* The subscriber might either continue to request more items every now and then or cut the connection by calling **Subscription::cancel**.