Overview of Popular Implementations of the Java Reactive Streams 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 the key benefits & principles underlying the reactive programming paradigm
- Know the Java reactive streams API & popular implementations of this API

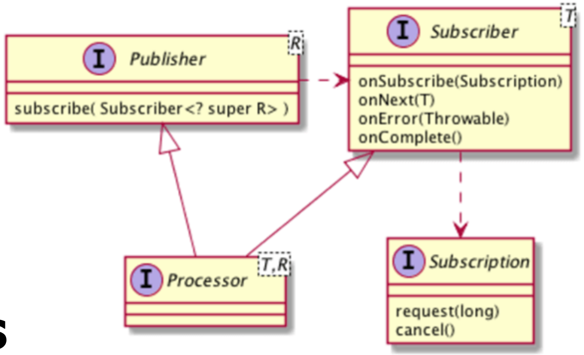
Project Reactor

See www.baeldung.com/rx-java & projectreactor.io

Popular Implementations of Java Reactive Streams

Popular Implementations of Java Reactive Streams

- The Java Flow API isn't very useful by itself



Useless Things

Popular Implementations of Java Reactive Streams

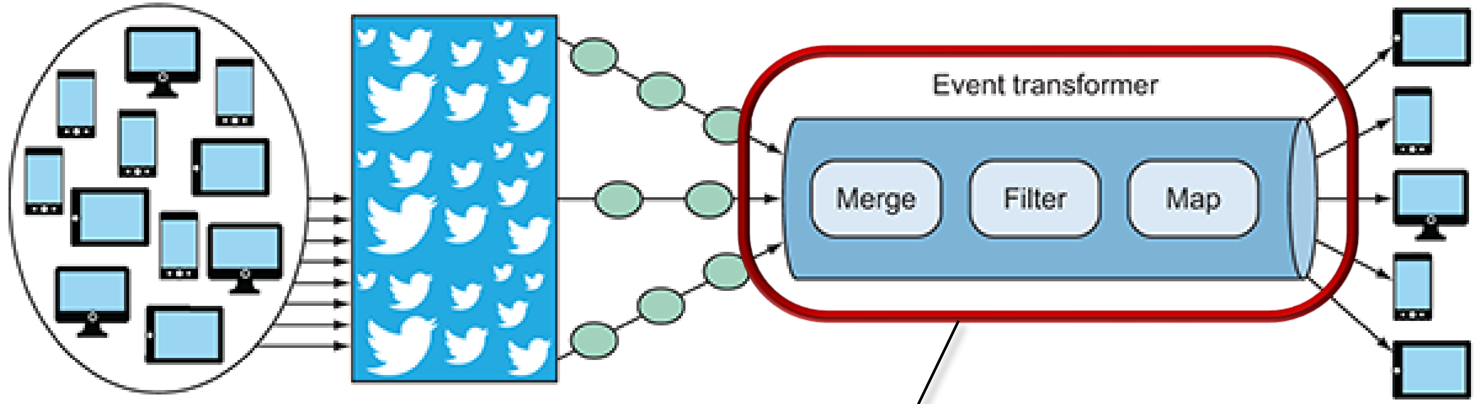
- The Java Flow API isn't very useful by itself
  - However, this API serves as an interoperable foundation implemented by other popular reactive programming frameworks

Project Reactor

See github.com/ReactiveX/RxJava/wiki & projectreactor.io

Popular Implementations of Java Reactive Streams

- Reactive streams implementations enable the insertion of event transformer operators between publishers & subscribers



Publishers

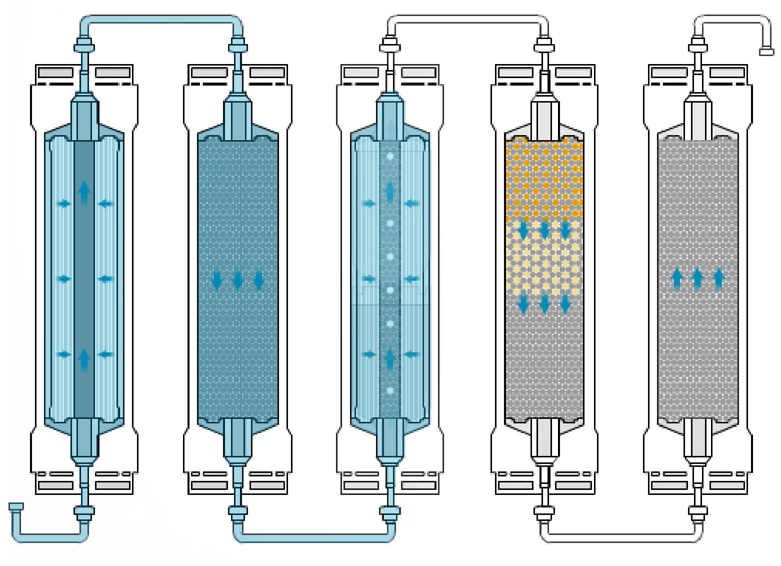
Subscribers

Data processed by a reactive app moves thru an "assembly line" (pipeline) of transformer operators

See projectreactor.io/docs/core/milestone/reference/#_from_imperative_to_reactive_programming

Popular Implementations of Java Reactive Streams

- Reactive streams implementations enable the insertion of event transformer operators between publishers & subscribers



Input x

Aggregate operation (behavior f)

Output f(x)

Aggregate operation (behavior g)

Output g(f(x))

Aggregate operation (behavior h)

Output h(g(f(x)))

Transformer operators are similar to aggregate operations in Java Streams

See docs.oracle.com/javase/tutorial/collections/streams

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

RxJava	Reactor	Purpose
Completable	N/A	Completes successfully or with failure, without emitting any value. Similar to Java CompletableFuture<Void>
Maybe<T>	Mono<T>	Completes successfully or with failure, may or may not emit a single value. Similar to an asynchronous Optional<T>
Single<T>	N/A	Either complete successfully emitting exactly one item or fails.
Observable<T>	N/A	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Does not support back-pressure due to the nature of the source of events it represents.
Flowable<T>	Flux<T>	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Supports backpressure (the source can be slowed down when the consumer cannot keep up)

See www.nurkiewicz.com/2019/02/rxjava-vs-reactor.html

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

RxJava	Reactor	Purpose
Completable	N/A	Completes successfully or with failure, without emitting any value. Similar to Java CompletableFuture<Void>
Maybe<T>	Mono<T>	Completes successfully or with failure, may or may not emit a single value. Similar to an asynchronous Optional<T>
Single<T>	N/A	Either complete successfully emitting exactly one item or fails.
Observable<T>	N/A	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Does not support back-pressure due to the nature of the source of events it represents.
Flowable<T>	Flux<T>	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Supports backpressure (the source can be slowed down when the consumer cannot keep up)

See github.com/ReactiveX/RxJava/wiki

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

RxJava	Reactor	Purpose
Completable	N/A	Completes successfully or with failure, without emitting any value. Similar to Java CompletableFuture<Void>
Maybe<T>	Mono<T>	Completes successfully or with failure, may or may not emit a single value. Similar to an asynchronous Optional<T>
Single<T>	N/A	Either complete successfully emitting exactly one item or fails.
Observable<T>	N/A	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Does not support back-pressure due to the nature of the source of events it represents.
Flowable<T>	Flux<T>	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Supports backpressure (the source can be slowed down when the consumer cannot keep up)

See projectreactor.io

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

RxJava	Reactor	Purpose
Completable	N/A	Completes successfully or with failure, without emitting any value. Similar to Java CompletableFuture<Void>
Maybe<T>	Mono<T>	Completes successfully or with failure, may or may not emit a single value. Similar to an asynchronous Optional<T>
Single<T>	N/A	Either complete successfully emitting exactly one item or fails.
Observable<T>	N/A	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Does not support back-pressure due to the nature of the source of events it represents.
Flowable<T>	Flux<T>	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Supports backpressure (the source can be slowed down when the consumer cannot keep up)

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

RxJava	Reactor	Purpose
Completable	N/A	Completes successfully or with failure, without emitting any value. Similar to Java CompletableFuture<Void>
Maybe<T>	Mono<T>	Completes successfully or with failure, may or may not emit a single value. Similar to an asynchronous Optional<T>

```
static Mono<Void> testFractionReductionSync() {
  ...
  return Mono
  .fromCallable(reduceFraction)
  .map(convertToMixedString)
  .doOnSuccess(printResult)
  .then(); ...
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/mono

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

RxJava	Reactor	Purpose
Completable	N/A	Completes successfully or with failure, without emitting any value. Similar to Java CompletableFuture<Void>
Maybe<T>	Mono<T>	Completes successfully or with failure, may or may not emit a single value. Similar to an asynchronous Optional<T>
Single<T>	N/A	Either complete successfully emitting exactly one item or fails.
Observable<T>	N/A	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Does not support back-pressure due to the nature of the source of events it represents.
Flowable<T>	Flux<T>	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Supports backpressure (the source can be slowed down when the consumer cannot keep up)

See projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

```
static <T> Flux<T> generate(Supplier<T> supplier,
                            long count) {
  return Flux
  .create(sink -> {
     LongStream.rangeClosed(1, count)
               .forEach(i -> sink.next(supplier.get()));
     sink.complete(); }); ...
```

RxJava	Reactor	Purpose
Flowable<T>	Flux<T>	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Supports backpressure (the source can be slowed down when the consumer cannot keep up)

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/flux

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

RxJava	Reactor	Purpose
Completable	N/A	Completes successfully or with failure, without emitting any value. Similar to Java CompletableFuture<Void>
Maybe<T>	Mono<T>	Completes successfully or with failure, may or may not emit a single value. Similar to an asynchronous Optional<T>
Single<T>	N/A	Either complete successfully emitting exactly one item or fails.
Observable<T>	N/A	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Does not support back-pressure due to the nature of the source of events it represents.
Flowable<T>	Flux<T>	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Supports backpressure (the source can be slowed down when the consumer cannot keep up)

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Single.html

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

RxJava	Reactor	Purpose
Completable	N/A	Completes successfully or with failure, without emitting any value. Similar to Java CompletableFuture<Void>
Maybe<T>	Mono<T>	Completes successfully or with failure, may or may not emit a single value. Similar to an asynchronous Optional<T>
Single<T>	N/A	Either complete successfully emitting exactly one item or fails.

```
static Completable testFractionMultiplicationCallable2(){ ...
  return Single
    .fromCallable(call)
    .subscribeOn(Schedulers.single())
    .doOnSuccess(bigFraction -> printResult(bigFraction, sb));
```

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Single

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

RxJava	Reactor	Purpose
Completable	N/A	Completes successfully or with failure, without emitting any value. Similar to Java CompletableFuture<Void>
Maybe<T>	Mono<T>	Completes successfully or with failure, may or may not emit a single value. Similar to an asynchronous Optional<T>
Single<T>	N/A	Either complete successfully emitting exactly one item or fails.
Observable<T>	N/A	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Does not support back-pressure due to the nature of the source of events it represents.
Flowable<T>	Flux<T>	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Supports backpressure (the source can be slowed down when the consumer cannot keep up)

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Observable.html

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

```
Observable.range(1, sMAX_FRACTIONS)
          .subscribe(__ -> emitter
                     .onNext(makeBigFraction(sRANDOM, false)),
                     t -> emitter.onComplete(),
                     emitter::onComplete);
```

RxJava	Reactor	Purpose
Observable<T>	N/A	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Does not support back-pressure due to the nature of the source of events it represents.
Flowable<T>	Flux<T>	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Supports backpressure (the source can be slowed down when the consumer cannot keep up)

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Observable

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

RxJava	Reactor	Purpose
Completable	N/A	Completes successfully or with failure, without emitting any value. Similar to Java CompletableFuture<Void>
Maybe<T>	Mono<T>	Completes successfully or with failure, may or may not emit a single value. Similar to an asynchronous Optional<T>
Single<T>	N/A	Either complete successfully emitting exactly one item or fails.
Observable<T>	N/A	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Does not support back-pressure due to the nature of the source of events it represents.
Flowable<T>	Flux<T>	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Supports backpressure (the source can be slowed down when the consumer cannot keep up)

See reactivex.io/RxJava/3.x/javadoc/io/reactivex/rxjava3/core/Flowable.html

Popular Implementations of Java Reactive Streams

- Reactive streams programs rarely use Publisher, Subscriber, & Subscription interfaces directly, but instead use classes that implement those interfaces

```
Flowable<Double> rateF = Flowable
        .just("GBP:USA")
        .parallel()
        .runOn(Schedulers.from(ForkJoinPool.commonPool()))
        .map(this::queryExchangeRateFor)
        .sequential()
        .timeout(2, TimeUnit.SECONDS, sDEFAULT_RATE_F);
```

RxJava	Reactor	Purpose
Flowable<T>	Flux<T>	Emits an indefinite number of events (zero to infinite), optionally completes successfully or with failure. Supports backpressure (the source can be slowed down when the consumer cannot keep up)

See github.com/douglascraigschmidt/LiveLessons/tree/master/Reactive/Flowable

End of Overview of Popular Implementations of the Java Reactive Streams API