Description

*Reactive programming* is programming with asynchronous data streams:

▶ Asynchronous like `CompleteableFuture`.
▶ Data streams like `Stream`.
Outline

General Concepts

Reactive Streams

Main Libraries

Summary and References
## Relation with Existing API

<table>
<thead>
<tr>
<th></th>
<th>single item</th>
<th>multiple items</th>
</tr>
</thead>
<tbody>
<tr>
<td>synchronous</td>
<td>$T$ data</td>
<td>$\text{Stream}&lt;T&gt;$ stream</td>
</tr>
<tr>
<td>asynchronous</td>
<td>$\text{CompletableFuture}&lt;T&gt;$ fut</td>
<td>$\text{Publisher}&lt;T&gt;$ pub</td>
</tr>
</tbody>
</table>

- A future/promise is to a value as publishers are to iterables/collections.
- Reacting to the completion of multiple futures (asynchronously, without blocking operation).
This is a click event represented by some value, e.g., a string.

This indicates the stream has completed.

This is an error.
Publish/Subscribe Model
Overflow Problem

- Too much *pressure* with push-based methods.
- Reactive programming is thus “pull-based” to control the pressure.
Asynchronous stream processing with non-blocking *backpressure*:
Other Characteristics

**Composable** chaining operations (like stream and completable future).

**Lazy evaluation** evaluated as late as possible (like stream).

**Asynchronous** non-blocking operations (like completable future).

**Reusable/Cacheable** results can be reused (like completable future).

**Push-based** the data source initiates the processing (pull-based for stream).

**Message passing** data producers exchange messages containing the data (no shared-memory mechanism).
History and Technological Context

- Kind of dataflow programming
- Also called FRP (Functional Reactive Programming): functional, no side effect, immutable state, pure function.
- Related to observer and iterator design patterns.
- Close to actor concurrency model.
Relation with Reactive Systems

- Reactive programming can be used to build a reactive system.
- The Reactive Manifesto states the features that must be offered by a reactive system: responsive, resilient, elastic, message-driven.
The system responds in a timely manner if at all possible. Responsiveness is the cornerstone of **usability**.

The system stays responsive in the face of **failure**.

The system stays responsive under **varying workload**. It can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs.

The system relies on **asynchronous message passing** to establish a boundary between components that ensures loose coupling, isolation, and location transparency.
Outlines

General Concepts

Reactive Streams
  Flow API (Java 9)
    SubmissionPublisher
    Implementation Example

Main Libraries

Summary and References
Main Interfaces

```java
interface Publisher<T> {
    void subscribe(Subscriber<T> subscriber);
}
interface Subscriber<T> {
    void onSubscribe(Subscription subscription);
    void onNext(T item);
    void onError(Throwable throwable);
    void onComplete();
}
interface Subscription {
    void request(long n);
    void cancel();
}
```
Publisher

subscribe

Subscriber

onSubscribe

Subscription

request(n)/cancel

onNext(data)

onError/onComplete

data

data
Reactive Streams Specification
Simplified Communication Flow

Subscriber

subscribe()

Publisher

new()

Subscription

onSubscribe(Subscription)

[loop]
[until completed]
request(int)

[alt]
[no error]
onNext(T)

[error]
onError(Throwables)

[alt]
[cancel subscription]
cancel()

onComplete()

Subscriber

Publisher

Subscription
## Dual to Iterable

<table>
<thead>
<tr>
<th>Event</th>
<th>Iterable (pull)</th>
<th>Observable (push)</th>
</tr>
</thead>
<tbody>
<tr>
<td>retrieve data</td>
<td><code>T next()</code></td>
<td><code>onNext(T)</code></td>
</tr>
<tr>
<td>discover error</td>
<td><code>throws Exception</code></td>
<td><code>onError(Exception)</code></td>
</tr>
<tr>
<td>complete</td>
<td><code>!hasNext()</code></td>
<td><code>onCompleted()</code></td>
</tr>
</tbody>
</table>
Outline

General Concepts

Reactive Streams
Flow API (Java 9)
SubmissionPublisher
Implementation Example

Main Libraries

Summary and References
Interface

Implementation of Flow.Publisher:

```java
SubmissionPublisher()
SubmissionPublisher(Executor executor,
                  int maxBufferCapacity);
CompletableFuture<Void> consume(Consumer<T> cons)
int submit(T item)
void close()
```
Example

```java
var pub = new SubmissionPublisher<Integer>();
pub.consume(System.out::println);
pub.submit(1);
```
[main] \rightarrow SubmissionPublisher

new() \rightarrow consume(Consumer) \rightarrow submit(Integer) \rightarrow Consumer.accept(Integer)

[main] \rightarrow SubmissionPublisher \rightarrow [executor]
Outline

General Concepts

Reactive Streams
- Flow API (Java 9)
- SubmissionPublisher
- Implementation Example

Main Libraries

Summary and References
class CustomSub implements Subscriber<Integer> {
    private Subscription subscription;
    public void onSubscribe(Subscription subscription) {
        this.subscription = subscription;
        subscription.request(1); } 
    public void onNext(Integer value) {
        System.out.println(value);
        subscription.request(1); } 
    public void onError(Throwables t) {
        System.err.println(t.getMessage()); } 
    public void onComplete() {
        System.out.println("Done!"); }
}
Chaining SubmissionPublisher

```java
var pub1 = new SubmissionPublisher<Integer>();
var pub2 = new SubmissionPublisher<Integer>();
var pub3 = new SubmissionPublisher<Double>();
var pub4 = new SubmissionPublisher<Double>();
pub1.consume(x -> {
    pub2.submit(x * x);
});
pub1.consume(x -> {
    pub3.submit(x / 2.);
});
pub2.consume(x -> {
    System.out.println("Square is: " + x);
});
pub3.consume(x -> {
    if (x > 2) pub4.submit(x);
});
pub4.consume(x -> {
    System.out.println("Half (> 2) is: " + x);
});
pub1.submit(3);
pub1.submit(5);
```
Outline

General Concepts

Reactive Streams

Main Libraries

Summary and References
Main Libraries

RxJava  Reactive Extensions, Netflix (more than 50 different operations)
Reactor  from Spring (equivalent to RxJava)
Akka  actor model
RxJava Example

```java
Flowable<Integer> flow = Flowable.range(1, 5)
    .map(v -> v * v)
    .filter(v -> v % 3 == 0)
    .subscribe(System.out::println);
```

- Flowable implements Publisher.
- subscribe triggers the execution (as with terminal operations, lazy evaluation).
Marble Diagrams

filter { }
Classic Operations

- map \{ \text{shape} \rightarrow \text{shape} \}
- allMatch(\text{shape})
- true
- skip(1)

- reduce \{ (\text{shape}, \text{shape}) \rightarrow \text{shape} \}
- exists
- false
- true
- limit(2)
Advanced Operations

- `collect`: 
  ```
  collect {{[diamond], [triangle]} -> [{diamond}]}
  ```

- `flatMap`: 
  ```
  flatMap { [red, green, blue] }
  ```

- `groupingBy`: 
  ```
  groupingBy
  ```

- `merge`: 
  ```
  merge
  ```

- `zip`: 
  ```
  zip {{cloud, triangle} -> {green}}
  ```
Concurrenty Source

- By default, the main thread performs all operations (blocking).
- The operation `subscribeOn` specifies how data processing can be processed concurrently.
- The operation at each stage can be performed asynchronously on a specific executor.
Outline

General Concepts

Reactive Streams

Main Libraries

Summary and References
Official Documentation

- Documentation of class Flow
- Documentation of class SubmissionPublisher
- Documentation of class Flowable
- Documentation of class Flux
To Go Further

- https://akarnokd.blogspot.com/
- https://github.com/reactive-streams/reactive-streams-jvm

Reactive Extensions (Rx) for Java or JavaScript:
- http://reactivex.io/
- https://github.com/ReactiveX/RxJava
- http://introtorx.com/
- https://gist.github.com/staltz/868e7e9bc2a7b8c1f754

Reactor:
- https://github.com/reactor/reactor-core
- https://spring.io/blog/2016/06/13/notes-on-reactive-programming-part-ii-writing-some-code
- https://spring.io/blog/2016/07/20/notes-on-reactive-programming-part-iii-a-simple-http-server-application
Other Sources

- http://alexsderkach.io/comparing-java-8-rxjava-reactor/
- https://spring.io/blog/2016/06/07/notes-on-reactive-programming-part-i-the-reactive-landscape
- https://grokonez.com/java/java-9-flow-api-reactive-streams
- https://www.futurice.com/blog/top-7-tips-for-rxjava-on-android/
- https://dzone.com/articles/5-things-to-know-about-reactive-programming