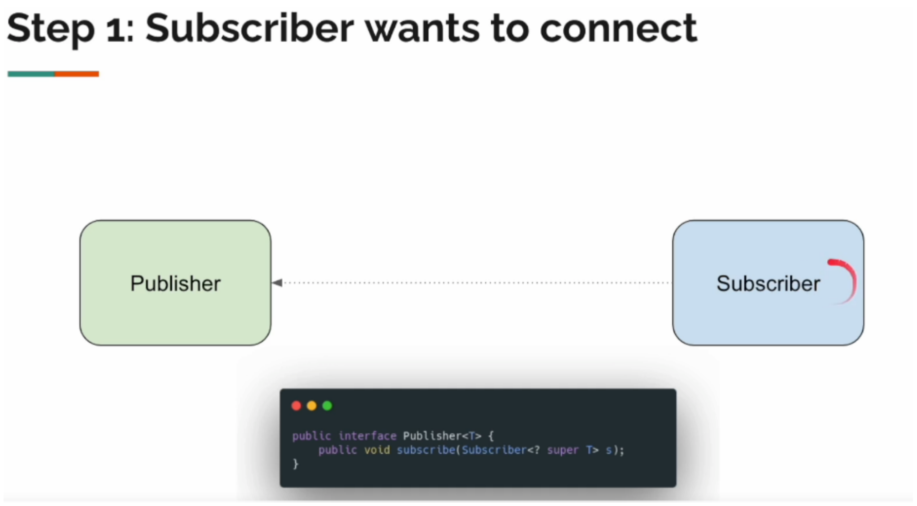
Publisher/Subscriber Communication

Wednesday 19 February 2025 16:02

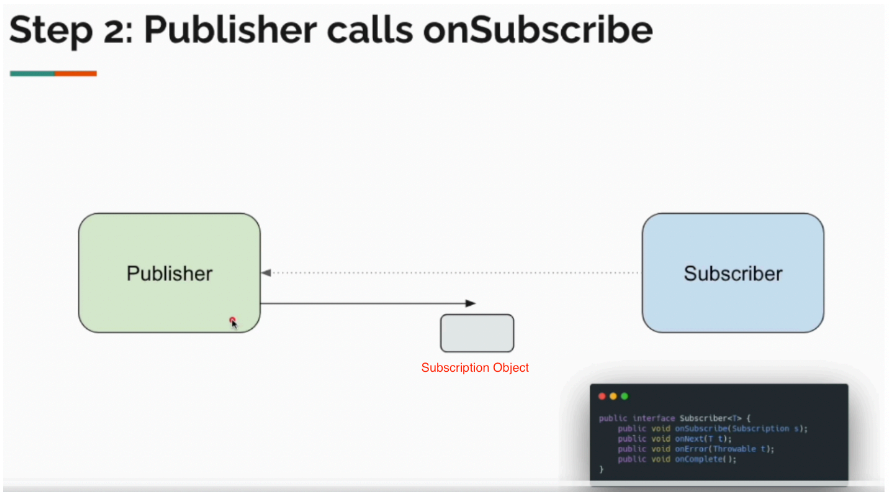
As we have understood that the Reactive Programming is based on the Publisher/Subscriber pattern. Now, it is time to understand some of the basic rules defined in the specification.



Step 1: Subscriber wants to connect

```
public interface Publisher<T> {
    public void subscribe(Subscriber<? super T> s);
}
```

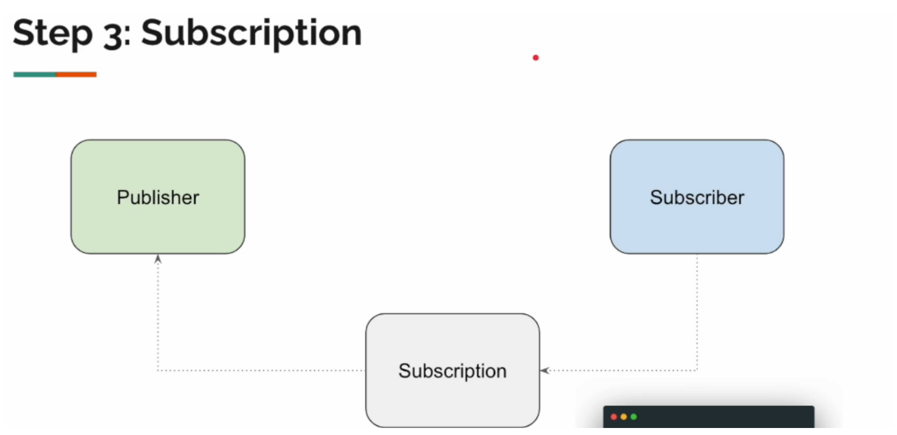
- There will be 2 instances. One will be publisher and the other one will be subscriber.
- The Subscriber wants to get updates from the Publisher
- The Publisher interface has the subscribe method which accepts the Subscriber instance. So using that we will be passing the Subscriber instance to the publisher to get the updates.



Step 2: Publisher calls onSubscribe

```
public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```

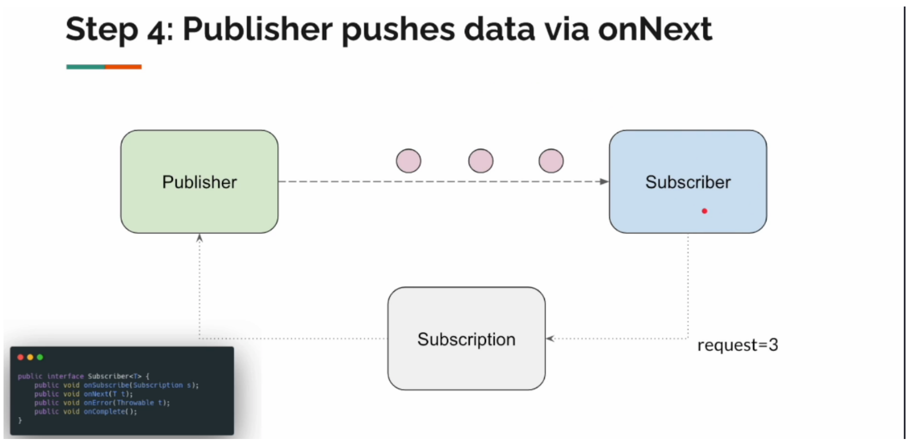
- When the Publisher accepts the request of the Subscriber, it hands over the Subscription object to the Subscriber.
- In the Subscriber interface, there is a onSubscribe method which accepts the Subscription object. So essentially the Publisher will give the Subscriber, the Subscription object using the onSubscribe method



Step 3: Subscription

```
public interface Subscription {
    public void request(long n);
    public void cancel();
}
```

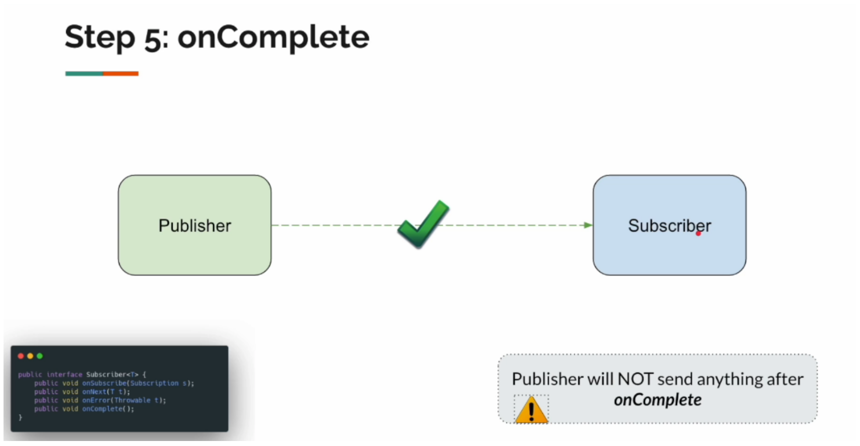
- The Publisher and Subscriber relationship is established at this point through the Subscription object.
- The Subscriber using the Subscription object can talk to the Publisher.
- The Subscriber can request for items (using the request method) from the Publisher to give items to the Subscriber.
- Or, if the Subscriber does not want to get any update from the Publisher, it can simply cancel the relationship (using the cancel method) with the help of the Subscription object.



Step 4: Publisher pushes data via onNext

```
public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```

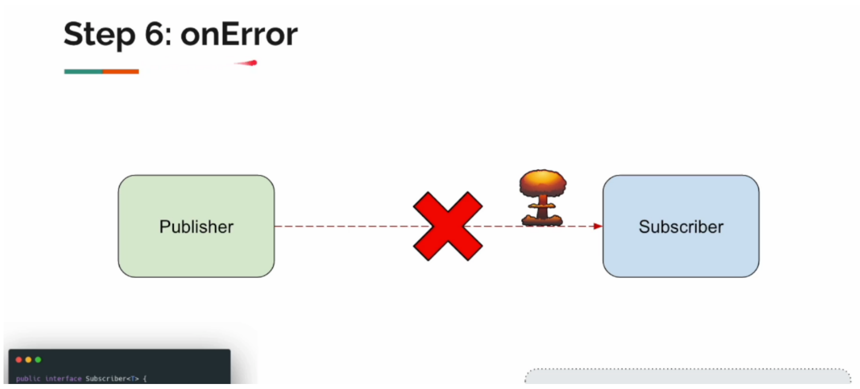
- When the Subscriber requests for some item to the Publisher using the Subscription object, the Publisher will use the onNext method of the Subscriber to pass the items to the Subscriber.
- So in this case the Subscriber has requested for 3 items. So the Publisher uses the onNext method to pass an item one by one sequentially to give the 3 items.
- IMPORTANT: The Publisher will only give the requested number of items or less. It will never give more.
- Using the Subscription object, the Subscriber can request for items again and again if it wants.



Step 5: onComplete

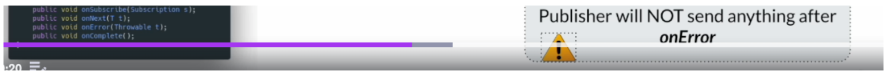
```
public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```

- When the Publisher has no item to emit, or If it has already emitted all the items to the Subscriber, the Publisher can call the Subscriber's onComplete method to notify the Subscriber that its job is done.
- As per the Reactive Stream Specification, at that point, once the onComplete method is invoked, then the Publisher will not be sending anything after that to the Subscriber. So we can see that their relationship ended at that point. Also, the Subscription object will no longer work.
- Then, the Subscriber cannot use the Subscription object to return more items
- So when you say that the Publisher has called onComplete, then the relationship of Publisher and Subscriber has ended at that point.



Step 6: onError

```
public interface Subscriber<T> {
```

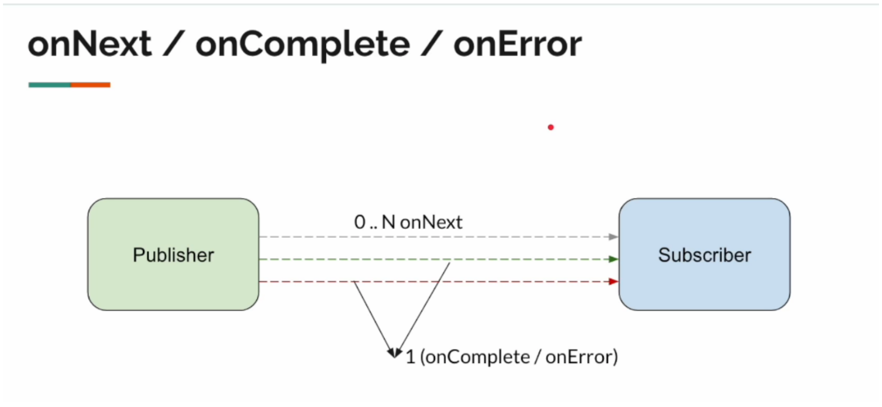


```
public void onSubscribe(Subscription s);
public void onNext(T t);
public void onError(Throwable t);
public void onComplete();
```

Publisher will NOT send anything after *onError*

- The Last step is onError
- As usual, the Subscriber will be requesting items from the Publisher using the Subscription object.
- While processing the request, the Publisher might face some issue.
- In that case, the Publisher will be passing the exception details to the Subscriber via the onError method.
- So, same as before, once you invoke the onError method, then the Publisher will not send anything after that to the Subscriber and the Subscription object will not work.

Quick Revision

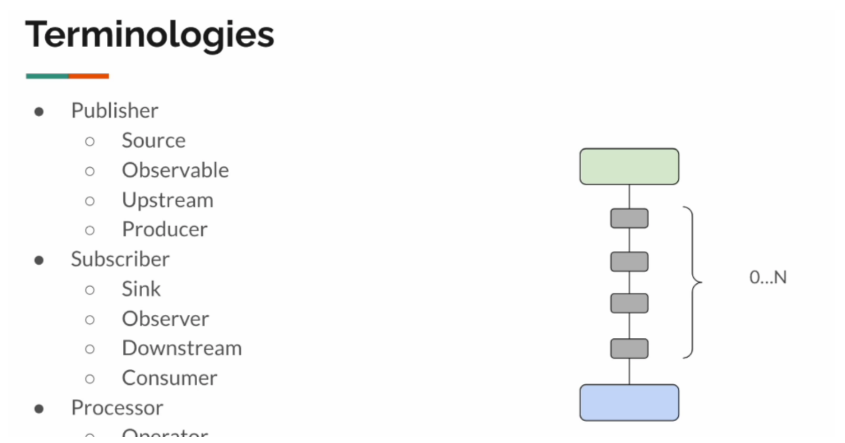


onNext / onComplete / onError

Lets quickly revise all the steps.

- We have a Publisher and we have a Subscriber.
- The Publisher will have a subscribe method using which we will be passing the subscriber instance to connect the publisher and the subscriber.
- So when the Subscriber subscribes to the Publisher, the Publisher will be passing one Subscription object to the Subscriber.
- Using this Subscription object, the Subscriber can request items from the Publisher.
- Based on the amount of items requested by the Subscriber, the Publisher will be giving items to the Subscriber using the onNext method
- The Publisher will emit only the requested number of items or less. It will not send more.
- So just because the Subscriber has requested for 100 items, the Publisher does not need to send 100 items. Only it has that many items, it will send them. Otherwise, it will not send.
- If it does not have any items to send, it can simply notify the subscriber saying that onComplete. So it says that I do not have anything to give you. So the Publisher can simply notify the Subscriber, whenever it does not have any items to give.
- So once the onComplete is invoked, the Subscriber cannot use the Subscription object to request items.
- On the other hand, while processing the request, when the Publisher encounters any issues, it will notify the Subscriber via the onError method. So even in this case, the Subscriber cannot use the Subscription object to request items after that.
- So the onComplete / onError method will be invoked only once. We can get either onComplete or onError not both.
- The onNext can be invoked multiple times, zero or more times, depending on the amount of data you are requesting and if the Publisher has it.

Terminologies

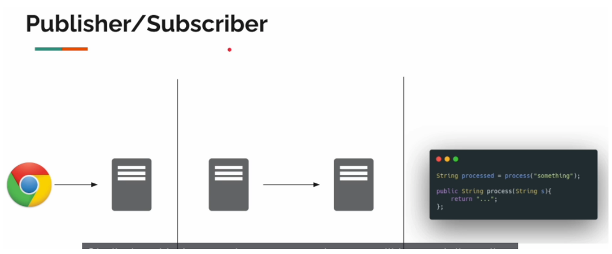


Terminologies

- Publisher
  - Source
  - Observable
  - Upstream
  - Producer
- Subscriber
  - Sink
  - Observer
  - Downstream
  - Consumer
- Processor

- Here we have a Publisher (in Green) and a Subsciber (in Blue).
- In between them, we can have 0 or more Processors.
- Depending on the usecase, we can have 0 Processors or more Processors.
- Sometimes, people can call the Publisher as Source, Observable, Upstream, Producer.
- Subscriber can also be called Sink, Observer, Downstream, Consumer, etc.
- The Processor, sometimes, can also be called as Operator.

In Reactive Programming, we will be modelling all the communication between the two components as Publisher and Subscriber



Publisher/Subscriber

```
String processed = process("something");

public String process(String s){
    return "...";
};
```

- For e.g., here we have a React Front End and REST API as backend. Then in that case, we can consider backend as a Publisher and the Frontend as the Subscriber because the FE expects data from the BE.
- Then in the 2nd example, you have a microservice architecture. One service can depend on another service. One can be Producer and other can be Subscriber.
- In the 3rd example, a class can have 2 methods. One method can depend upon another method. In this example, the process method can act like a Producer and the statement with processed variable can act like a Subscriber.
- So anything that give data can act like a Producer and anything that asks data acts like a Subscriber.