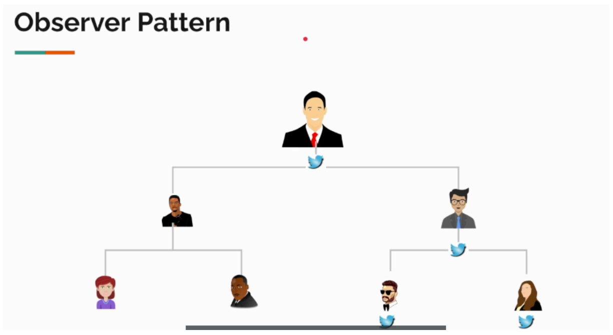
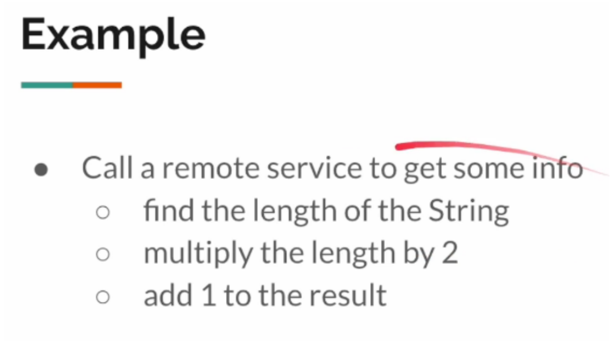
Reactive Stream Specification

Tuesday 4 February 2025 16:24


Observer Pattern

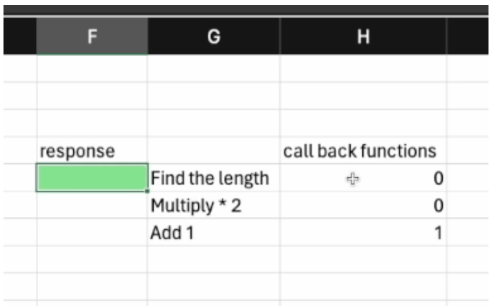
- Reactive programming is based on the observer pattern.
- One example is Twitter.
    - People follow each other on twitter.
    - Lets say on the top is a famous actor (lets call him X) and lets say the person on the middle right (lets call him Y) follows him.
    - Whenever X tweets, Y can see.
    - If Y reacts to the message by commenting or re-tweeting, then the followers of Y can also see.
    - This is what we call the observer pattern on a higher level - that is, to observe and react in case of changes

Lets take another example



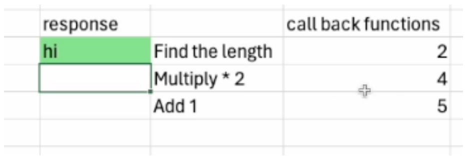
Example

- Call a remote service to get some info
    - find the length of the String
    - multiply the length by 2
    - add 1 to the result

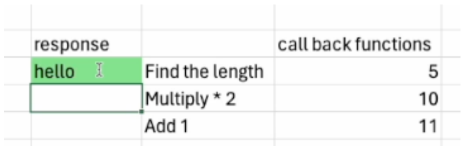
- Lets consider this business requirement.
- We have to call the remote service to get some information. That information can be simple, for example, we can say that it is a string. Now our complex business requirement is
    - Find the length of the string
    - Multiply the length by 2
    - Add 1 to the result
- 

	F	G	H
	response		call back functions
	🟩	Find the length	0
		Multiply * 2	0
		Add 1	1

- We can use this excel sheet to understand how the observer pattern works, and how the reactive programming works on a high level.
- Now lets consider our business requirement, we have to call the remote service in a non-blocking manner, so the OS will notify when the response is received.
- So lets imagine that the OS will populate the green cell when the response comes back.
- Now I have to use or provide my complex business logic in a callback functions like find the length, multiply * 2, add 1, etc.
- Here, although we use excel formula, we get the overall idea.
- So the cell H5 (right to the 'Find the length' - G5) observes the green box F5. Whatever the value is In F5, find the length of that.
- H6 overserves H5, that is, what is the result of finding the length, multiply it by 2
- Similarly, H7 observes H6, that is, we add 1 to the result of H6.
- So this is our complex business logic, we have provided everything as a callback function.
- So the OS will notify me when we get the response, could be after 1 secs or 5 secs.

- 

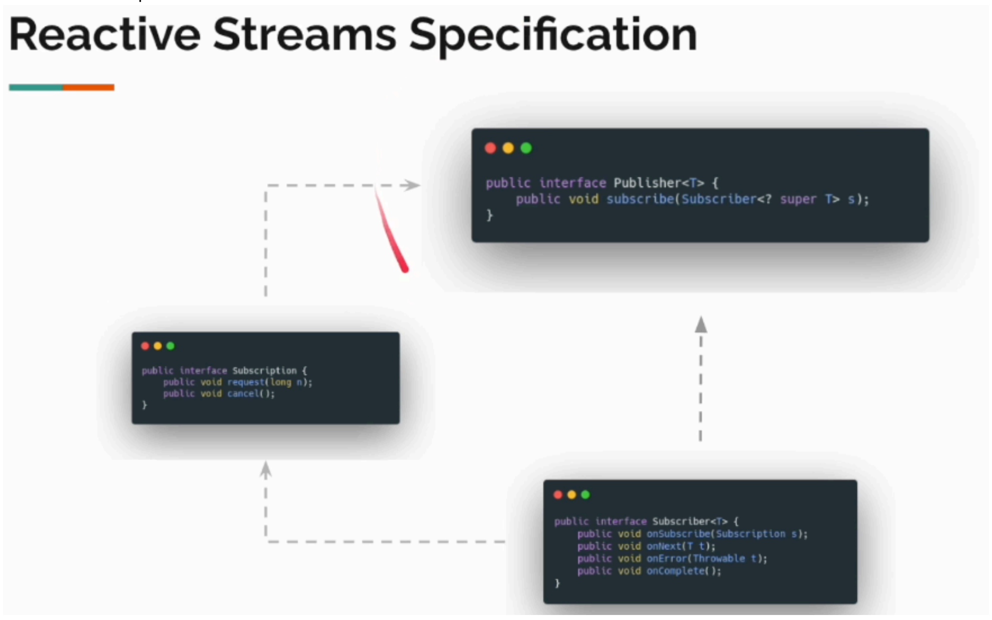
response		call back functions
hi	Find the length	2
	Multiply * 2	4
	Add 1	5

- Now as you see above, as soon as we get the response, we immediately react to the data and get the result.
- 

response		call back functions
hello	Find the length	5
	Multiply * 2	10
	Add 1	11

- Lets say if we get a different result, we immediately react. So this is what is reactive programming i.e. observe and react to the stream of messages.

Reactive Stream Specification

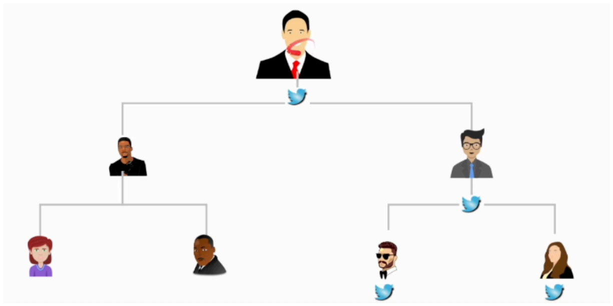


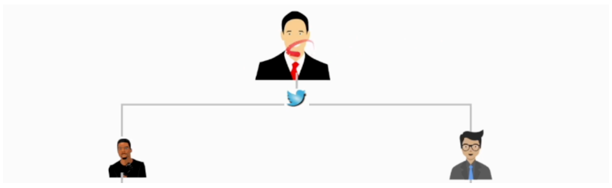
Reactive Streams Specification

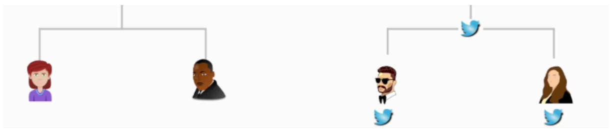
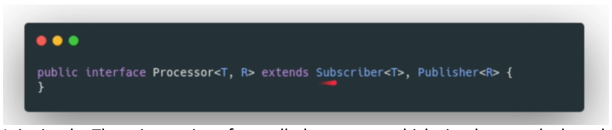
```
public interface Publisher<T> {
    public void subscribe(Subscriber<? super T> s);
}
```

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

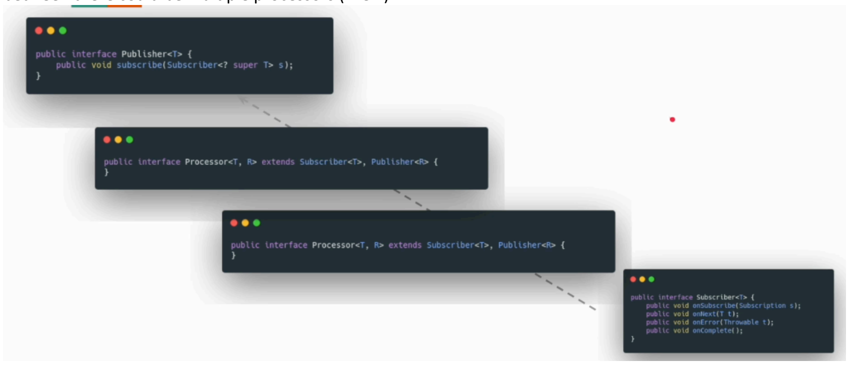
```
public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}
```

- As part of the reactive stream specification, they have defined a set of interfaces to model how the communication will work.
- 
- There will be a publisher. If we take the above twitter example where Y (where Y is the person on the middle right) follows X (where X is the person on the top) to get his tweets
- So X will be the publisher and Y will be the subscriber. X follows Y to get his tweets.
- So the subscriber follows the publisher and their relationship is displayed via an subscription object.
- If the subscriber want to get more updates from the publisher (or in this case wants to see more tweets), he can do so by using the request method, it can always request to the publisher.
- Similarly, if the subscriber doesn't want to get any updates from the publisher (or he wants to unfollow the publisher), then it can cancel the relationship using the cancel method.
- If we see the above picture with interfaces, these are all simple interfaces, just a specification, not a library, which we could directly use.



- 
- In the twitter example, Y was the subscriber to X, so that Y can get X's tweets.
- But Y can have its own followers. So if Y does something the followers can observe.
- So how to represent Y in the reactive stream specification where Y is both publisher and subscriber.
- 

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

- It is simple. There is new interface called processor which simply extends the subscriber and publisher. So the processor can act both like a subscriber and publisher.
- So Y acts a subscriber to X and acts a publisher to its own subscribers.
- There would be a publisher at the top (say X), and there could a subscriber at the bottom, but in between there could be multiple processors (like Y).
- 

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

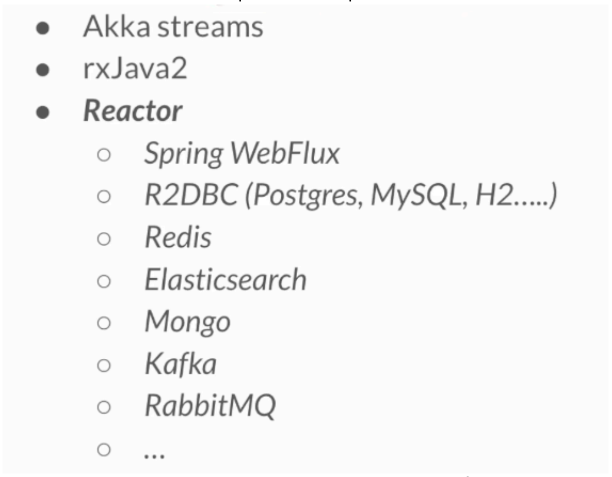
```java
public interface Processor<T, R> extends Subscriber<T>, Publisher<R> {
}
```

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

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

- This is how it will look like. There will be a publisher and there could be multiple processors, and finally there would be a subscriber.

Reactive Streams - implementation

- We have JPA as a specification which is implemented by Hibernate among others.
- Similarly, Reactive Streams is only a specification.
- There are some libraries which provide the implementation for us.
- 

  - Akka streams
  - rxJava2
  - ***Reactor***
    - *Spring WebFlux*
    - *R2DBC (Postgres, MySQL, H2.....)*
    - *Redis*
    - *Elasticsearch*
    - *Mongo*
    - *Kafka*
    - *RabbitMQ*
    - ...

- In this course, we are going to use Reactor. We have support from Spring Team. Then we have relational database driver called R2DBC (which supports Postgres, MySQL, H2, etc), Redis, Elasticsearch, Mongo Driver, Kafka, RabbitMQ, etc.