RxJS (Observables & Subject) – Async Processing of data

Pub/Sub Design Pattern (Publisher [producers] – Subscribers [Consumers])

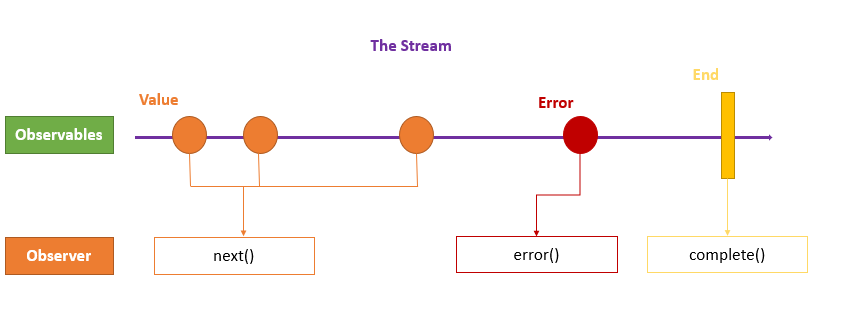
HttpClient – This is used to get the data from API (webservice) and update the Angular UI accordingly

EventEmitters – Which Emit and manage the Events

**Observables**

Observables provide support for passing messages between parts of your application. We can use observables for

* event handling,
* asynchronous programming, and (non-blocking operations)
* handling multiple values.

In earlier version of angular, we used promises to handle async data.

Now a days, we use Observable to handle async data.

Difference between Promises & Observables is Promises emits a Single value but Observables emits multiple value.

Observables will have 3 call back methods namely next(), error() & complete().

To call these three methods of Observables, first we need to subscribe to an Observable

Example:

import {Observable, of } from "rxjs";

export class AppComponent {

// Create simple observable that emits three values

myObservable = of(1, 2, 3);

// Create observer object

myObserver = {

next: x => console.log('Observer got a next value: ' + x),

error: err => console.error('Observer got an error: ' + err),

complete: () => console.log('Observer got a complete notification'),

};

// Execute with the observer object

constructor(){

this.myObservable.subscribe(this.myObserver);

}

// Logs:

// Observer got a next value: 1

// Observer got a next value: 2

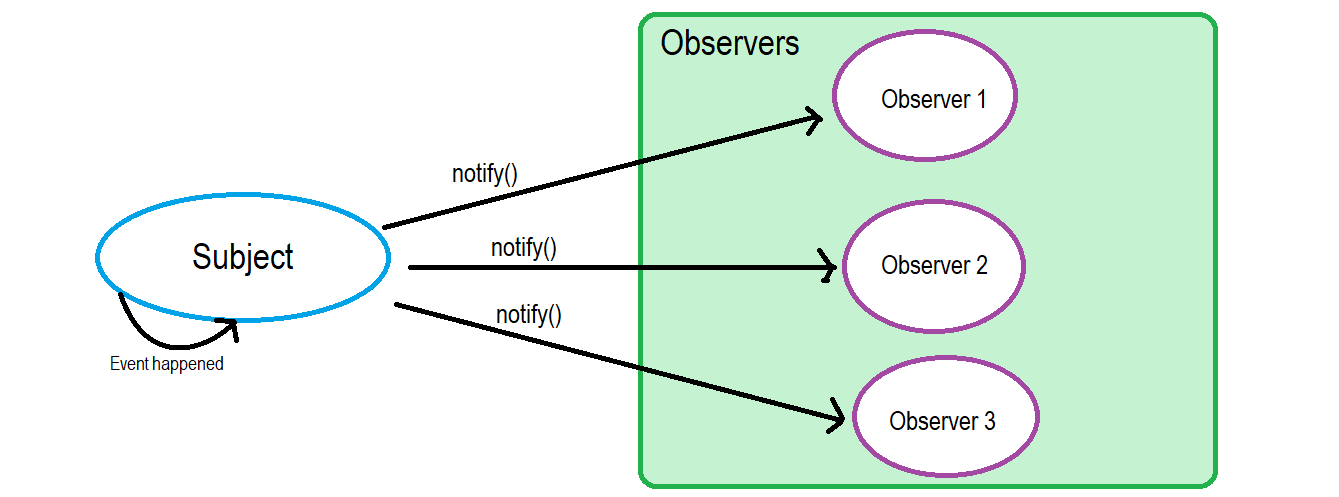
// Observer got a next value: 3

// Observer got a complete notification

}

RxJS = **R**eactive e**x**tension for **J**ava **S**cript

Subjects



Types of Subject

1. behavior subject
2. Replay subject
3. Async subject

EventEmitters

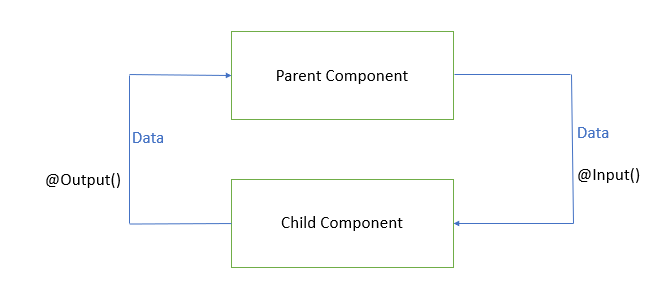
Events are nothing but some operation/action in the UI (user interface) click, doubleclick, keypress, mouseover, pageload, pagereload, submit, blur

An [EventEmitter](https://angular.io/api/core/EventEmitter" \t "_blank) is used to emit custom events synchronously or asynchronously, and register handlers for those events by subscribing to an instance.

Decorators used for transferring data between components

@Input

@Output



1. App-Module (Is the container of all the components) – Root Module
2. App-Component (Is the root component of Angular )
3. User-defined components

<https://www.split.io/blog/crud-app-spring-boot-angular/>

<https://www.bezkoder.com/spring-boot-angular-13-crud/>