Angular Observables(13)

https://ide.c9.io/laczor/angular
session_13_observables-start
160-162 Custom Observable, Unsubscribe, More info
session_13_observables-Final
*** 163 Subject ***(coding into the files!)
*** 164 Observable operators ***

160-163 Custom Observable, Unsubscribe, More info

www.reactivex.io - more information, about the observables

home.component.ts

```
// 1. Basically, we are importing the Observalbe + the Rx package, so we can create observalbes
// 2. Observables, are basically a information or information package, which can be emitted, fa
iled, complteded, any assychron totally
// 3. We are setting a constant to be an observable, and emitting it's value, 1 second, then wh
e are subscribeing to this observable, so we can executed some code when we are recieving the d
ata
// 4. easy to see and implement usage, of sharing data.


import { Component, OnInit,OnDestroy } from '@angular/core';
import {Observable} from 'rxjs/Observable';
import {Observer} from 'rxjs/Observer';
import {Subscription } from 'rxjs/Subscription';


import 'rxjs/Rx';                          // To work with observable operators.
@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit, OnDestroy {
// numbersObsSubscription : Subscription;
 myObservableObsSubscription : Subscription;
```

```
  constructor() { }


  ngOnInit() {


 // ******Lecture 160 ******
 // 1. We are manually creating an osbervable, with observer functions. So what we define, is t
hat we have a datapackage, what we would like to send, then we are determining
//  in the observer methods, in which way, and what sould be emited, also what will be the type
 of the emitted data package,
// 2. So when we are subscribing to the observable, we will know, how to handle if the emitting
 (Provides data/ Fail/ completes)
    const myObservable = Observable.create( (observer: Observer<string>)=>{
        setTimeout(()=>{
          observer.next('First package');
        },2000);
        setTimeout(()=>{
          observer.next('Second package');
        },4000);
        setTimeout(()=>{
          observer.error('error has been occured');
        },5000);
        setTimeout(()=>{
          observer.complete();
        },7000);
        setTimeout(()=>{
          observer.next('Last package');
        },8000);
    });
  this.myObservableObsSubscription = myObservable.subscribe(
    (data:string)=>{console.log(data);},                                          //1. data recieved
    (error:string)=>{console.log(error);},                                        //2. Error occured
    ()=>{console.log('completed argument reache for the observable');}      //3. observer compl
eted
    );


  }
// **** Lecture 161 UNSUBSCRIBE ****
// ---home.component.ts--
// 1. We are importing, and implementing ngOnDestroy, interfaces, lifecyclehook, so when the co
mponent is destroyed, we can executed some of the codes
// 2. Then importing subscription type as well, so we can create a custom property of the compo
nent, which will store our subscription, and when the component is destroyed
//  with the built in unbsubscribe method, we can quickly terminate it
// 3. Really, important, because it affects the memory capacity,performance, information leak.


    ngOnDestroy(){
      this.myObservableObsSubscription.unsubscribe();
    }
}
```

*** 163 Subject ***

This is an observable + an observer at the same time, can emit and subscribe to itself!
Has to import the code, but it is a good alternative of emitting events.
So we created a new service, which actually, just creates a new subject, and in the components where we are injecting it, we are using the subject's built in functions
like emiiting data, + subscribing to it in an other component.

1. Create a **user.service.ts** which will create a new subject

```
import { Subject } from 'rxjs/Subject';


export class UsersService {
  userActivated = new Subject();
}
```

2. register it at the **app.module.ts**

```
import { UsersService } from './users.service';

  providers: [UsersService],
```

3. Use the service and send a subject package from **user.component.ts**

```
import { Component, OnInit } from '@angular/core';

import { ActivatedRoute, Params } from '@angular/router';

import { UsersService } from '../users.service';                        //Import the previously
 created service


@Component({

  selector: 'app-user',

  templateUrl: './user.component.html',

  styleUrls: ['./user.component.css']

})

export class UserComponent implements OnInit {

  id: number;

  constructor(private route: ActivatedRoute, private usersService: UsersService) { }        //I
njecting the service


  ngOnInit() {

    this.route.params

      .subscribe(

        (params: Params) => {

          this.id = +params['id'];

        }

      );

  }


  onActivate() {
    this.usersService.userActivated.next(this.id);                               //Emittin
g data
  }
}
```

4. Listening to the emitted data by the subject service at **app.component.ts**

```
import { Component, OnInit } from '@angular/core';
import { UsersService } from './users.service';                        //Import user srevice

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  user1Activated = false;
  user2Activated = false;

  constructor(private usersService: UsersService) {}                //Inject it
                                                                    //Subscribe for the emittin
g datapackages
  ngOnInit() {
    this.usersService.userActivated.subscribe(
      (id: number) => {
        if (id === 1) {
          this.user1Activated = true;
        } else if (id === 2) {
          this.user2Activated = true;
        }
      }
    );
  }
  }
```

*** 164 Observable operators ***

- can modify the value of the observable, within the observalbes
- Can be applied to any observable.
map operator:--> (maps the data what we get and maps it to a new observable with every required modification)