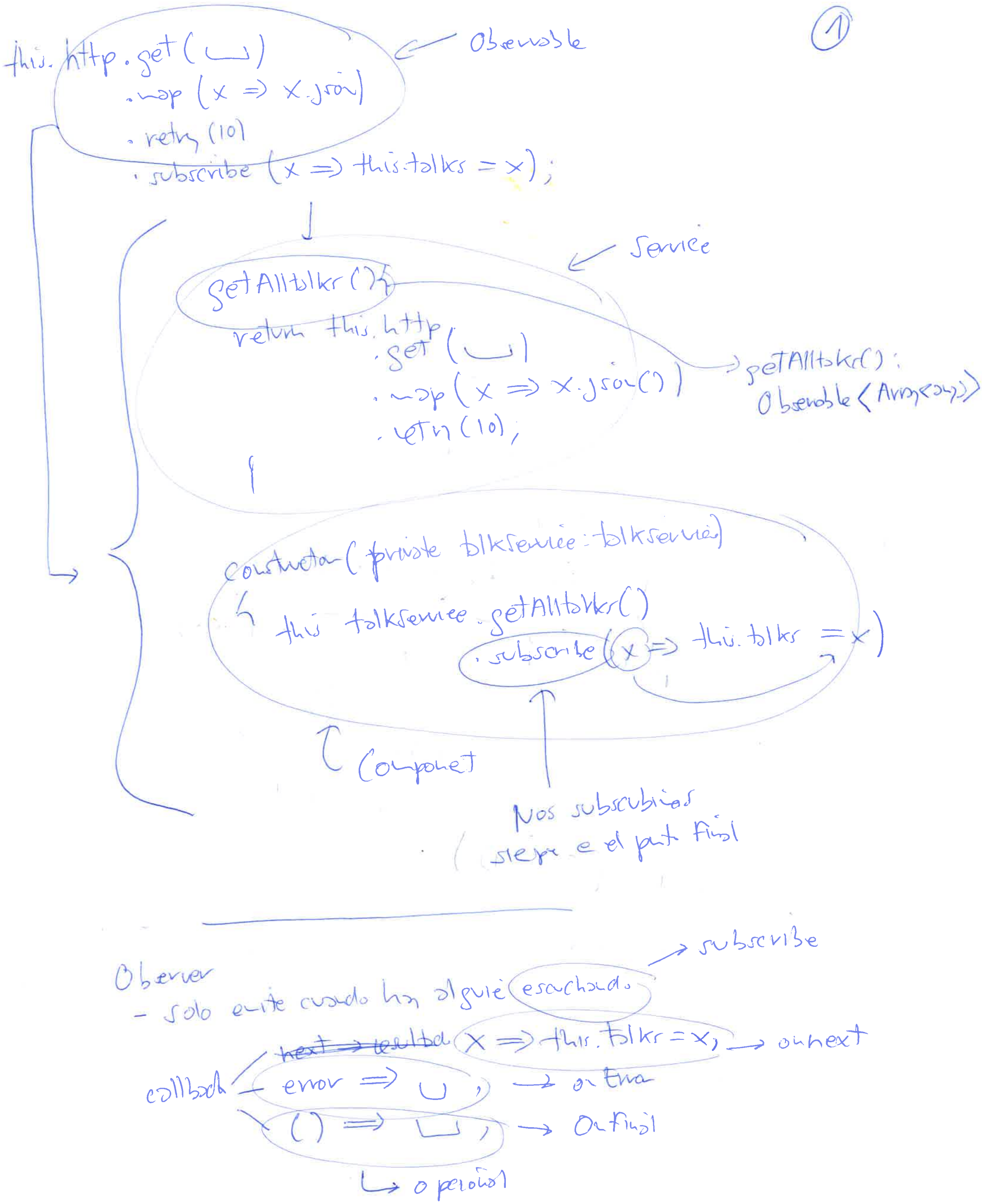
this.http.get(...)
.map(x => x.json)
.retry(10)
.subscribe(x => this.talks = x);
Observable
1
Service
getAlltalkr():
Observable<Array>
getAlltalkr(){
return this.http
.get(...)
.map(x => x.json())
.retry(10);
}
constructor(private talkservice:talkservice)
{
this.talkservice.getAlltalkr()
.subscribe(x => this.talks = x)
}
Componet
Nos subscribimos
siempre al put final
Observer
- solo existe cuando hay alguien escuchando.
subscribe
callback
next → resultado
x => this.talkr = x;
onNext
error => ...,
onError
() => ...,
onFinal
o periodo

- Eventos múltiples

```
Observable.interval(1000)
    .subscribe( x => console.log(x) );
```

- Operadores
  map, filter, etc...

- Combinación

```
let obs1 = ...
let obs2 = ...
return obs1.merge(obs2);
```

- schedulers.

Rx básico

A) http.get → Devuelve Observable

B) Observable.of(2,1,2)

```
Observable.of(...[2, 3, 4]).subscribe( x => console.log(x) );
```

(...) ← operador spread

2
3
4

```
.map( x => 10 * x)
```

20
30
40

```
.filter( x => x > 100)
```

Va al final

c) Observable.from([2,3,4])

→ Diagramas de marble.

Usando Rx

<input ... #search> ← local template variable

```
@ViewChild('search') search: ElementRef;

ngOnInit{
    console.log(this.search)
}
```

ElementRef → nativeElement → DOM

nativeElement ← varía dependiendo de quién lo ejecute:
- navegador
- app iOS
- app android

Observable.fromEvent(this.search.nativeElement, 'keyup')
.subscribe(x => console.log(x))
.map((x:any) => console.log(x.target.value))

Arrow function

.filter(x => x.length > 2)

Espera el último fuera

.debounceTime(500)

↑ solo envía evento cada x tiempo

.distinctUntilChanged

↑ filtra repetidos

.switchMap(x => this.talkService.getFilteredTalks(x))

.subscribe(x => this.talks = x)

↳ Filtrado de cambios por búsqueda

↘ Cancela petición cuando aparece otra.

http

nx Angular	Operadores habituales

Observable.create

range → devuelve un

interval

epth

throw

creación

transformación

buffer/bundle

groupBy

map

FlatMap

Solo devuelve un evento

Devuelve cualquier cosa de resultado

Aplica a cualquier

Filtrodi

debounce
distinct
filter
first
sample
skip
take

Combinazioni

combineLatest
concat
merge
zip
race
startwith
switch

de errori

catch
retry
retryWhen

de utilità

do → per fare logs.
delay
materialize
timeout
toArray
topromise

Rx Avanzato

Multicast < cache, publish, share

publish → obs.connect

obs.subscribe
obs.subscr.

cold observable / hot observable → cache, publish

~~cache~~

Subject < Observable ← Entrata / Observer ← Emite

BehaviorSubject ultimo

AsyncSubject

unsubscribe → ng OnDestroy

Async pipe

this.take = Se-
{ bind.
| async

~~subscribe~~