
User:

```
import { Observable } from 'rxjs'; const observable = new Observable(observer => {
  observer.next(Math.random()); // Emits a random number
  observer.complete(); }); // Each subscription gets
a separate execution
observable.subscribe(value => console.log('Subscriber 1:', value));
observable.subscribe(value => console.log('Subscriber 2:', value));
```

In the above code it is very much clear that which is subscriber, observable and observer but what will be in the case of HTTP request?

ChatGPT:

In the case of an HTTP request using RxJS, the `Observable` wraps the HTTP request, and each subscription triggers a new execution of that request. Here's how it works:

Mapping to HTTP Requests:

- **Observable** → Represents the HTTP request.
- **Observer** → Listens for the response, error, or completion.
- **Subscriber** → Calls `subscribe()` to trigger the request.

Example Using RxJS with an HTTP Request:

```
typescript
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

// Assume this is in an Angular service
class ApiService {
  constructor(private http: HttpClient) {}

  fetchData(): Observable<any> {
    return this.http.get('https://api.example.com/data');
  }
}

const apiService = new ApiService(new HttpClient(/* Injector dependency */));

// Each subscription triggers a new HTTP request
apiService.fetchData().subscribe(data => console.log('Subscriber 1:', data));
apiService.fetchData().subscribe(data => console.log('Subscriber 2:', data));
```

Key Takeaways:

1. **Separate Executions:** Just like in your example where each subscription gets a new random number, here, each subscription triggers a new HTTP request.
2. **Cold Observable:** HTTP requests using `HttpClient` in Angular return cold observables, meaning the request is not sent until `.subscribe()` is called.
3. **To Share Response:** If you want to share the same response among multiple subscribers, use `shareReplay()` to cache the result:

```
typescript
import { shareReplay } from 'rxjs/operators';
```

```
const sharedObservable = apiService.fetchData().pipe(shareReplay(1));

sharedObservable.subscribe(data => console.log('Subscriber 1:', data));
sharedObservable.subscribe(data => console.log('Subscriber 2:', data));
```

This ensures the HTTP request is made only once and the response is shared.

Would you like further clarification on RxJS behaviors with HTTP? 🚀

User:

I guess in http request the observer just listens the response but in new Observable it emits the value as well with next().

ChatGPT:

Yes, you got it! The key difference is how the **observer** behaves in each case:

1 Manually Created Observable (new Observable())

- The **observer** is responsible for **emitting** (next()), **erroring** (error()), and **completing** (complete()).
- The execution logic is defined inside the Observable constructor.
- Each subscriber gets a fresh execution.

Example:

```
typescript
import { Observable } from 'rxjs';

const observable = new Observable(observer => {
  observer.next(Math.random()); // Emits a random number
  observer.complete();
});

observable.subscribe(value => console.log('Subscriber 1:', value));
observable.subscribe(value => console.log('Subscriber 2:', value));
```

- Here, the observer calls next(), which actively **emits** values.
- Each subscriber gets a separate random number.

2 HTTP Request (HttpClient.get())

- The **observer does NOT emit** values manually.
- Instead, the HttpClient.get() **internally handles** making the request and pushing the response when available.
- The observable is "cold," meaning the request isn't sent until someone subscribes.

Example:

```
typescript
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

class ApiService {
  constructor(private http: HttpClient) {}

  fetchData(): Observable<any> {
    return this.http.get('https://api.example.com/data'); // No manual `next()`
  }
}

const apiService = new ApiService(new HttpClient(/* injector */));

apiService.fetchData().subscribe(data => console.log('Subscriber 1:', data));
apiService.fetchData().subscribe(data => console.log('Subscriber 2:', data));
```

- Here, the observer **only listens** for the response but does not control emissions.
- Every subscription triggers a **new HTTP request** (unless we use operators like `shareReplay()`).

🔥 Key Difference:

Feature	<code>new Observable()</code>	<code>HttpClient.get()</code>
Who emits?	Observer calls <code>next()</code>	HTTP request emits when response is received
Execution	Defined in constructor	Handled internally
Multiple Subscribers	New execution each time	New request each time (unless shared)

Would you like an example with `shareReplay()` to reuse the same HTTP response? 🚀