Programación Reactiva con Rx js

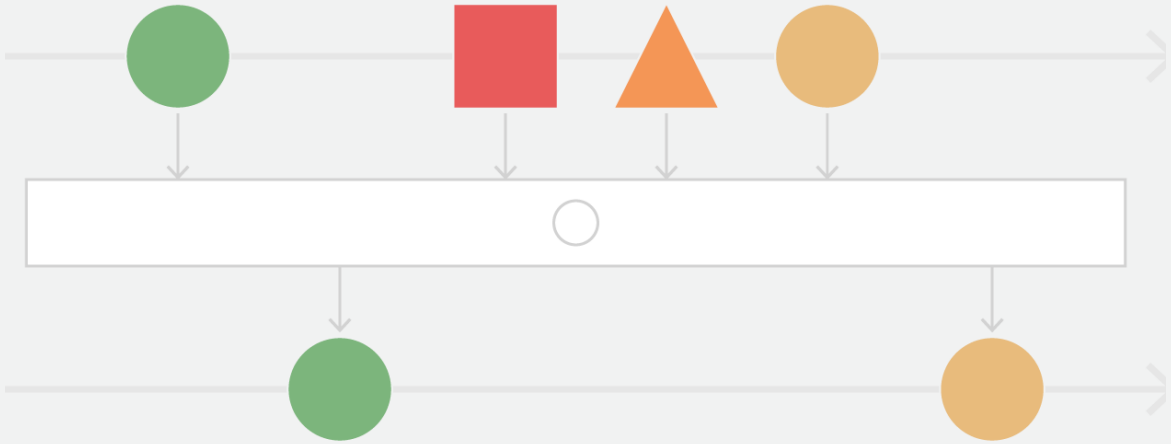
¿Qué es la programación reactiva?

Es una forma de construir aplicaciones utilizando EVENTOS y REACCIONANDO a ellos, estos eventos pueden ser compuestos, filtrados ordenados, etc. Algunos ejemplos:

- En el Front End cuando configuramos Listeners para eventos de usuarios (onClick, mouseOver, etc).
- En el Back End cuando configuramos métodos de respuesta a peticiones del front end, etc.

¿Cómo funciona?

En la programación reactiva todos los datos fluyen como una corriente de agua, esta corriente es una *"secuencia de eventos"* que representan valores, errores o eventos completados, esta corriente puede ser escuchada, filtrada, mezclada e incluso podemos crear una nueva corriente de datos a partir de ella.



Para interactuar con esta corriente debemos "Suscribirnos" a ella definiendo un Listener que sea capaz de manejar sus tres posibilidades.

- Un listener se denomina "Observer".
- Una corriente de datos se llama "Observable".

Observables

Un observable es parecido a un Array, un array es una colección de valores y un observable le añade el concepto de valores a traves del tiempo, es decir: mientras que un array obtiene sus valores de una vez, los observables obtiene sus parámetros al pasar el tiempo incluso pasado minutos.

Y, ¿Qué tiene que ver todo esto con Angular?

¿Cómo utiliza Angular la programación reactiva?

RxJs

RxJS es una librería para construir programas asincrónicos y basados en eventos mediante secuencias observables.

Proporciona un tipo de núcleo, el Observable, tipos de satélite (Observer, Schedulers, Subjects) y operadores inspirados en Array#extras (map, filter, reduce, every, etc) para permitir el manejo de eventos asincrónicos como colecciones.

Los conceptos esenciales en RxJS que resuelven la gestión de eventos asíncronos son:

- Observable: representa la idea de una colección invocable de valores o eventos futuros.
- Observer: es una colección de devoluciones de llamada que sabe escuchar los valores entregados por el Observable.
- Subscription: representa la ejecución de un Observable, es principalmente útil para cancelar la ejecución.
- Operators: son funciones puras que permiten un estilo de programación funcional para el manejo de colecciones con operaciones como: map, filter, merge, etc.
- Subject: es el equivalente a un EventEmitter, y la única forma de multidifusión de un valor o evento a múltiples observadores.
- Schedulers: son despachadores centralizados para controlar la concurrencia, lo que nos permite coordinar cuando el cálculo ocurre.

Volviendo a los Observables ...

Comparando un array con un observables:

Si queremos operar cada valor de una Array utilizamos el operador map, luego si queremos filtrar el array utilizamos el operador filter, para finalmente imprimir los resultados con el operador forEach.

```
[1, 2, 3, 4, 5]
  .map(x => x * 2)
  .filter(x => x > 5)
  .forEach(x => console.log(x)); // 6, 8, 10
```

Volviendo a los Observables ...

Si queremos hacer lo mismo con Observables:

Importamos los módulos necesarios de rxjs

```
import { from } from 'rxjs';
import { filter, map } from 'rxjs/operators';
```

Y utilizamos los mismos operadores sobre el observable.

```
from([1, 2, 3, 4, 5])
  .pipe(
    map(x => x * 2),
    filter(x => x > 5)
  )
  .subscribe(x => console.log(x)); // 6, 8, 10
```

¿Y como trabajar con Rx js y Angular?