# Implementar Observables

Para trabajar con Observables vamos hacer uso de tres actores:

* **Un *subject:*** es el encargado de generar y emitir los eventos de actualización del almacén de datos
* **Un observable:** que es un objete que permite observar los eventos emitidos por el subject
* **Una suscripción:** que se genera a partir del observable.

**Nota:** otras implementaciones pueden poner en juego otros tipos de objetos.

**El servicio va a ser el encargado de utilizar el *subject* para generar los eventos cuando se agreguen nuevos elementos.**

**Este servicio también será el encargado de entregar el *observable* a los componentes que necesiten escuchar los eventos emitidos por el *subject***

**Aquellos componentes que van a escuchar los eventos son los observadores (*observers)***. **Estos observadores declaran y utilizan el *observable* y mediante una *suscripción* se van a enteran de cualquier cambio del *observable***

## Producir el flujo de eventos a observar

El servicio que implementa toda la parte de acceso a los datos es el encargado de generar los eventos que los observadores van a consumir. Por lo cual el servicio debe conocer la implementación de dos clases:

1. El subject: para genera el stream de eventos
2. El observable: que va a entregar bajo la demanda de los observadores

Para cual el servicio va necesitar los siguientes *Imports*



**Nota:** no tenemos que instalar nada ya que la librería RxJS se instaló cuando creamos el proyecto de Angular.

## Declaración del subject como propiedad

El *subject* será u elemento del servicio por lo cual lo debemos declarar como una propiedad del mismo.



* Declaramos al *subject* como privado para que nadie pueda acceder salvo el mismo servicio.
* Por convención al nombre de los subject se le suele colocar el signo $ al final para dejar en claro lo que son.
* Utilizamos un *generic* de TypeScript para indicar aquello que vamos a observar: en este caso lo que vamos a observar es un array de clientes por eso en la instanciación del *subject* se define como <*Cliente[]>*
* No hace falta indicar el tipo de la propiedad *nombreVariable: Subject* dado que se infiere debido a que se instancia en la misma línea que la declaración de la propiedad

## Emitir eventos usando el *subject*

Cada vez que se agregue un nuevo elemento al array de clientes tenemos que emitir un evento utilizando el *subject*

Así quedaría el método agregar un cliente: antes solo realizábamos el *push()* y ahora agregamos la generación del evento con el *subject*

El método que vamos a utilizar del *subject* es *next* al que le debemos pasar como parámetro el array para que los observadores puedan saber cómo estaba el array al producirse este evento



## Generar el observable

Por último, vamos a generar el *observable* que se entregara a todos aquellos componentes que quieran observar cambios en el almacén de datos.

El *observable* es un consumidor de los eventos del *subject* y es de solo lectura, es decir va a estar atento a los eventos, pero no hace nada más.

Mediante el *observable* los componentes sabrán cuando el almacén de datos se ha modificado, pero ningún componente podrá generar nuevos eventos de cambio del almacén de datos

El *observer* se va a crear mediante un método del *subject* llamado *asObservable().* Vamos a crear un método *getter* para devolver el observable:



El método devuelve un objeto de la clase *observable*, el cual también se debe especificar utilizando un genérico de TypeScript declarando que ese observable es capaz de observar un array de Clientes []

## Consumir un observable

Ahora en cualquier componente que necesitemos que este atento a los cambios del almacén de datos podemos usar el observable que nos ofrece el servicio, para hacerlo debemos hacer lo siguiente:

En el componente importar la clase *observable*



Luego vamos a crear un atributo dentro de la clase de nuestro componente para almacenar el observable



Por ultimo vamos a crear la suscripción a los eventos que nos entrega el observable y que fueron generados en el servicio utilizando el subject

El lugar más adecuado para generar la suscripción es el ngOnInit() dado que se va a ejecutar cuando el componente ya se haya inicializado y por lo tanto tiene todas sus propiedades disponibles.

La suscripción la hacemos mediante el método *subscribe* del observable. Este método recibe como parámetro que va ser la función que se va a llamar cuando se dispare el evento. Esta función manejadora de eventos recibe como parámetro el array que se está observando



**Nota:** en el ejemplo utilizamos la función flecha para definir la función del manejador de eventos de la suscripción, pero podríamos haber usado una función anónima.  
Es recomendable utilizar una función flecha por dos motivos:

* La sintaxis es más reducida y concisa
* La función flecha no genera un contexto propio, por lo cual podemos seguir utilizando *this*.

Básicamente lo que hace la función es que cada vez que cambia el array de clientes, se actualiza la propiedad interna del componente al array actualizado (que es recibido como parámetro)

# Resumen: pasos para utilizar observables:

## Para generar el observable

1. En el servicio que se encarga del acceso a los datos vamos a importar dos clases de RxJS: *subject* y *observable*



1. Dentro del servicio creamos una propiedad del tipo *subject* y la inicializamos con genéricos de TypeScript con el tipo de elementos del almacén de datos



1. Dentro de los métodos del servicio que modifican el almacén de datos emitimos el evento con el método *next* del *subject* y pasamos como parametro el listado de datos

this.clientes$.next(this.clientes);

1. Dentro del servicio generamos un método que devuelva un observable también utilizando genéricos con el tipo de datos del almacén. El método va a devolver lo que devuelve el método asObservable() del *subject*



## Para consumir el observable

1. Importamos la clase *observable* de la librería RxJS



1. Creamos un atributo del tipo observable dentro del componente que implemente generics del tipo de datos del almacén



1. En el ngOnInit() del componente nos suscribimos al observable

