ANGULAR

COMUNICACIONES ENTRE COMPONENTES

LOS COMPONENTES NO SON AISLADOS

- Nunca lo han sido
- Los “componentes” de HTML se comunican entre ellos, o con nosotros, en ambos sentidos
- De entrada:

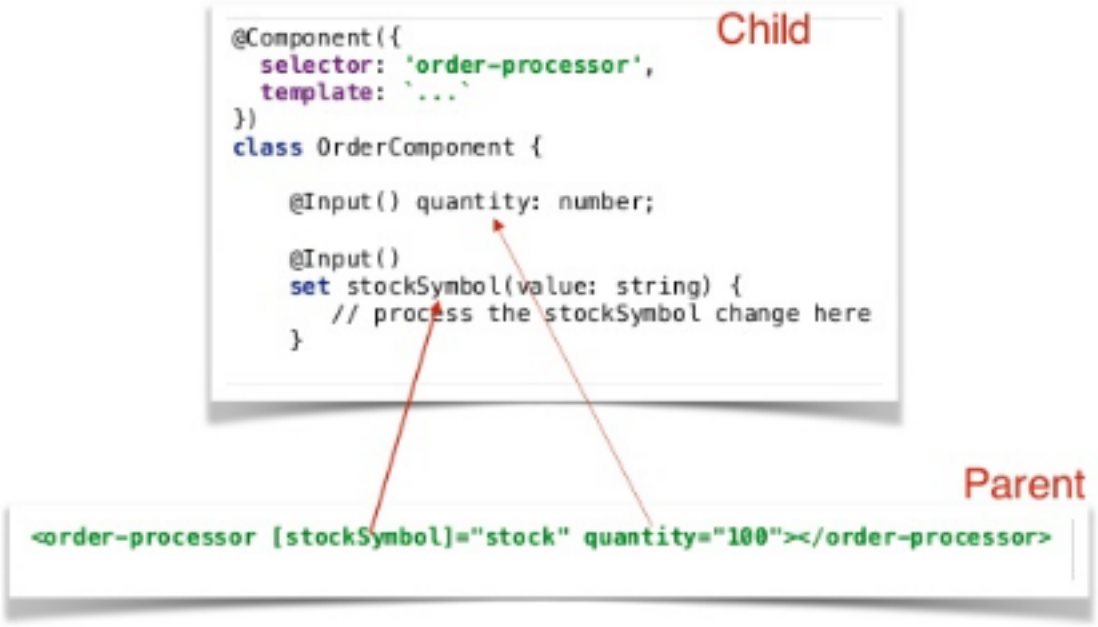
```
▼<div class="field-wrapper">
   <label for="username" class>Username</label>
   <input type="text" name="username" id="username" tabindex="1" autofocus="autofocus" autocorrect="off"
   autocapitalize="off" class=" av-text" value>
  </div>
```

- Y de salida:

```
▼<form name="login" id="login" class="login" method="post" action="/booked/Web/index.php">
```

LOS COMPONENTES NO SON AISLADOS

- Del mismo modo, nuestros componentes se pueden comunicar en ambos sentidos
  - El sentido de entrada será un one-way databinding desde la vista que tenga nuestro componente, hacia el controlador de nuestro componente (@Input)
    - [property]="valor"
    - property="valor"



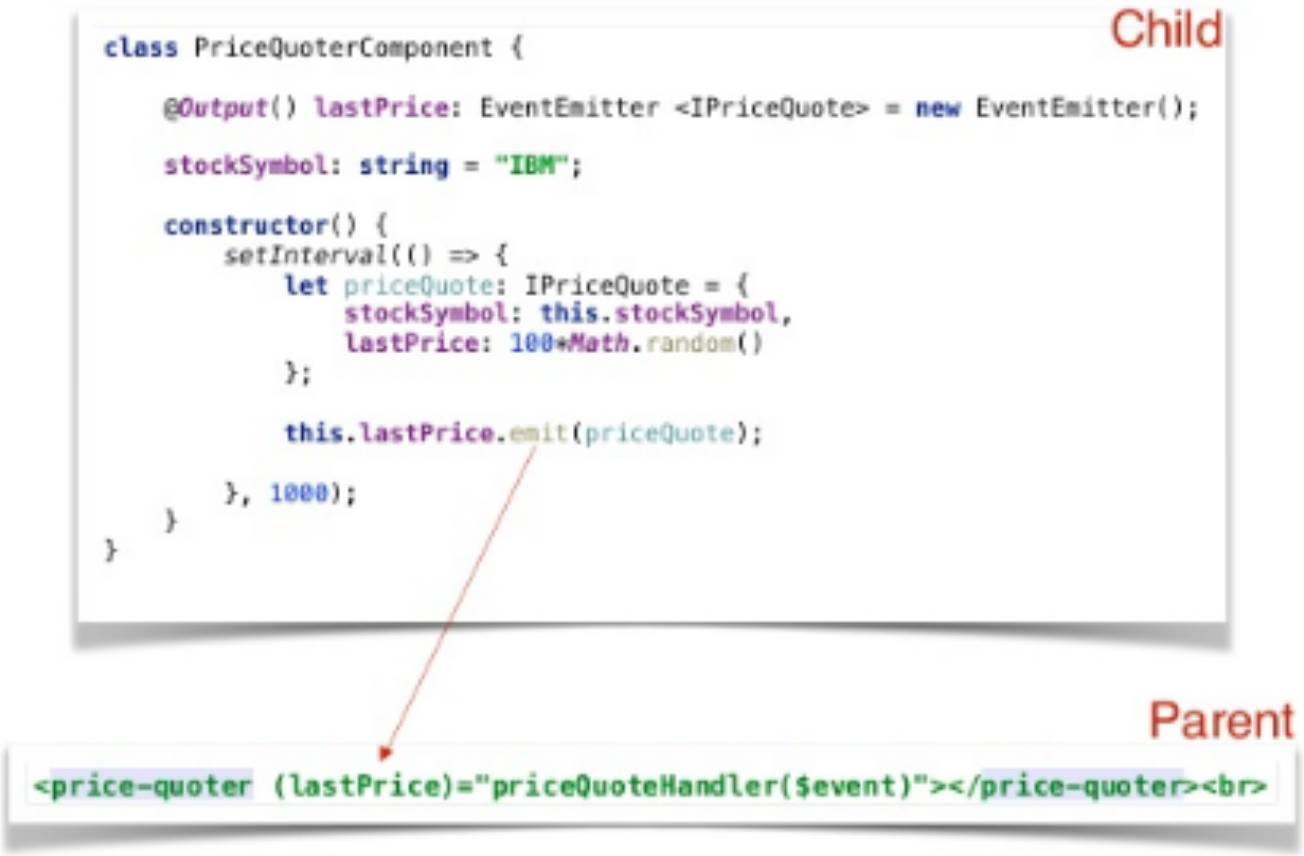
INPUT

- El @Input no tiene mucho misterio
- Si ponemos @Input es una propiedad de la clase de nuestro controlador, simplemente estamos abriendo a que el valor de esa propiedad sea “rellenado” por otro controlador, que llamaremos padre.
  - En este caso se referencia como nombrePropiedad=“valor”
- Si por el contrario, el @Input se lo establecemos a un setter, significa que queremos hacer alguna acción en el momento en el que se establece ese valor desde fuera
  - En este caso se referencia como [nombrePropiedad]=“valor”
- Por supuesto, estos valores pueden venir de propiedades de otros controladores, y ser rellenadas en el momento que se precise
  - E incluso, pueden ser Observables…
- Se puede pasar como parámetros al @Input el nombre por el que queramos que se referencia este fuera de la clase. De este modo podemos tener el nombre interno que queramos, y exponer un nombre más amigable al exterior

```
@Input() ciudad: string;
@Input('pais') nacionReal: string;
```

LOS COMPONENTES NO SON AISLADOS

- Mientras que el sentido de salida será un one-way databinding desde el controlador, a la vista que esté tratando el componente (@Output)
  - (event)="myEvent"



OUTPUT

- @Output es algo más complejo
- Siempre se utiliza junto a un objeto de la clase EventEmitter
  - Este objeto nos permitirá "pushear" valores que serán recogidos por todos aquellos que estén "escuchando" a ese evento
  - Como todo en TypeScript, podemos tipar fuertemente ese EventEmitter para forzar a que emita siempre valores del tipo que nosotros queramos

```
@Output() onVoted = new EventEmitter<boolean>();
```

- Cuando nuestro componente estime oportuno, emitirá valores de nuestro @Output para que quienquiera que esté al otro lado (nuestro componente padre) haga lo que crea oportuno. Esa emisión siempre es a través de push(…)

```
vote(agreed: boolean) {
  this.onVoted.emit(agreed);
  this.voted = true;
}
```

OUTPUT

- Ahora ya estamos fuera de nuestro componente. Estamos en el componente padre, que va a incluir al nuestro
- Siguiendo el esquema anterior, esta sería la forma en la que el componente padre se "suscribiría" al evento que acabamos de exponer

```
<my-voter *ngFor="let voter of voters"
  [name]="voter"
  (onVoted)="onVoted($event)">
</my-voter>
```

- La función de callback onVoted podría ser así

```
onVoted(agreed: boolean) {
  agreed ? this.agreed++ : this.disagreed++;
}
```

HIJO, YA NO HABLAMOS...

HAY OTRAS FORMAS DE COMUNICARSE...

COMUNICACIÓN HIJO – PADRE POR VARIABLE TEMPORAL

- Hasta ahora habíamos dicho que un componente exponía todos sus métodos públicos a los demás
- Como hemos visto, no es suficiente con esto…
- En realidad, la relación de herencia en Angular tiene un componente inverso
- Es el padre el que puede acceder a las propiedades y métodos del hijo, siempre y cuando éstos sean públicos
- Esto se consigue a través de las variables locales en template
- Es mejor verlo con un ejemplo:

COMUNICACIÓN HIJO – PADRE POR VARIABLE TEMPORAL

- Dado el siguiente componente:

```
export class CountdownTimerComponent implements OnInit,
OnDestroy {

  intervalId = 0;
  message = '';
  seconds = 11;

  clearTimer() { clearInterval(this.intervalId); }

  ngOnInit()    { this.start(); }
  ngOnDestroy() { this.clearTimer(); }
```

COMUNICACIÓN HIJO – PADRE POR VARIABLE TEMPORAL

- El componente padre puede acceder a sus propiedades públicas así:

```
<h3>Countdown to Liftoff (via local variable)</h3>
<button (click)="timer.start()">Start</button>
<button (click)="timer.stop()">Stop</button>
<div class="seconds">{{timer.seconds}}</div>
<countdown-timer #timer></countdown-timer>
```

- Con el símbolo “#” definimos una variable local de nombre “timer” que tiene el valor de countdown-timer
  - Si os ayuda, el valor de this
- Por tanto, podemos usar nuestra flamante variable timer para llamar a los métodos y propiedades públicas de countdown-timer
- Como se puede apreciar, el orden de declaración es irrelevante

COMUNICACIÓN HIJO – PADRE POR VARIABLE TEMPORAL

COMUNICACIÓN HIJO – PADRE POR VIEWCHILD

- Las variables temporales tienen una limitación
  - Es la vista del padre el que debe saber las propiedades del hijo, y escribirlas correctamente
  - No hay comprobación de tipos, ni control por parte del padre
  - Todas las propiedades públicas del hijo quedan expuestas, sin remedio
- Para controlar todos estos aspectos, se puede utilizar un método similar al anterior, pero que en lugar de enlazarse a la plantilla directamente, se enlaza al controlador del padre a través de la anotación @ViewChild

```
@ViewChild(CountdownTimerComponent)
private timerComponent: CountdownTimerComponent;
```

COMUNICACIÓN HIJO – PADRE POR VIEWCHILD

- La inicialización de variables, o la enmascaración de las mismas, se debe hacer en el evento adecuado ngAfterViewInit:

```
  seconds() { return 0; }

  ngAfterViewInit() {
    // Redefine el valor de seconds(), que será
obtenido a través del componente hijo
`CountdownTimerComponent.seconds` ...
    // pero tenemos que esperar un tick
primero, porque se suelen producir condiciones
de carrera
    // en los flujos de datos unidireccionales
    setTimeout(() => this.seconds = () =>
this.timerComponent.seconds, 0);
  }
```

https://angular.io/api/core/testing/tick

COMUNICACIÓN ENTRE COMPONENTES POR SERVICIO

- Quizá una de las formas más naturales para los desarrolladores de comunicarnos entre componentes sea a través de un servicio
- Pero como hemos podido ver, la comunicación entre componentes a través de un servicio no es tan simple como parece
- Por el propio ciclo de vida de un servicio, un componente, sus componentes hijos, etc., cuando requerimos un dato compartido, no tiene porqué estar disponible.
  - Es más, por el propio flow de datos de Angular, aunque luego el dato esté disponible, no habrá forma de que nos enteremos de ello…
- Para comunicarnos a través de servicios necesitamos MARCIANIZAR nuestros servicios…

COMUNICACIÓN ENTRE COMPONENTES POR SERVICIO

- Quizá una de las formas más naturales para los desarrolladores de comunicarnos entre componentes sea a través de un servicio
- Pero como hemos podido ver, la comunicación entre componentes a través de un servicio no es tan simple como parece
- Por el propio ciclo de vida de un servicio, un componente, sus componentes hijos, etc., cuando requerimos un dato compartido, no tiene porqué estar disponible.
  - Es más, por el propio flow de datos de Angular, aunque luego el dato esté disponible, no habrá forma de que nos enteremos de ello…
- Para comunicarnos a través de servicios necesitamos MARCIANIZAR nuestros servicios…

COMUNICACIÓN ENTRE COMPONENTES POR SERVICIO

```
@Injectable()
export class MissionService {

  // Observable string sources
  private missionAnnouncedSource = new
Subject<string>();
  private missionConfirmedSource = new
Subject<string>();

  // Observable string streams
  missionAnnounced$ =
this.missionAnnouncedSource.asObservable();
  missionConfirmed$ =
this.missionConfirmedSource.asObservable();
```

COMUNICACIÓN ENTRE COMPONENTES POR SERVICIO

```
  // Service message commands
  announceMission(mission: string) {
    this.missionAnnouncedSource.next(mission);
  }

  confirmMission(astronaut: string) {

this.missionConfirmedSource.next(astronaut);
  }
}
```

Esto es todo amigos

MUCHAS GRACIAS