AMPLIANDO CONOCIMIENTOS DE COMPONENTES

ÍNDICE

- Tipos de relaciones entre componentes
- Relación a través del decorador @Input()
- Creando eventos con @Output()
- Relación mediante el uso de servicios
- Utilizando NgClass y NgStyle
- Directiva ngSwitch
- Operaciones especiales, los *pipes*

TIPOS DE RELACIONES ENTRE COMPONENTES

```
<div id="contenedor">
  <h3>Listado de entradas al blog:</h3>
  <app-entrada *ngFor="let entrada of listadoEntradas" [entrada]="entrada" (
click)="mostrarTitulo(entrada.titulo)"></app-entrada>
</div>
```

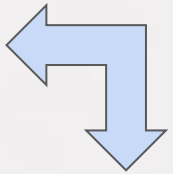
PADRE-HIJO

Cuando hablamos de relación Padre-Hijo entre componentes nos referimos a cuando la etiqueta de un componente se encuentra en la plantilla de otro.

TIPOS DE RELACIONES ENTRE COMPONENTES

- Comunicación Padre-Hijo a través del decorador *@Input()*
- Comunicación Padre-Hijo con *ngOnChanges()*
- Comunicación Hijo-Padre a través de eventos
- Acceso del componente Padre a atributos y métodos del componente Hijo
- Acceso del componente padre a atributos y métodos del componente Hijo a través del decorador *@ViewChild()*
- Comunicación entre componentes a través de un servicio

```
<div id="contenedor">
  <h3>Listado de entradas al blog:</h3>
  <app-entrada *ngFor="let entrada of listadoEntradas" [entrada]="entrada" (
click)="mostrarTitulo(entrada.titulo)"></app-entrada>
</div>
```



@Input()

Se enlaza el atributo de la etiqueta con el atributo del componente

```
export class EntradaComponent implements OnInit {
  // Atributos
  @Input()
  public entrada: Entrada;

  constructor() {
    this.entrada = {
      titulo: '',
      resumen: ''
    }
  }

  ngOnInit(): void {
  }

}
```

```
export class EntradaComponent implements OnInit, OnChanges {
  // Atributos
  @Input()
  public entrada: Entrada;

  constructor() {
    this.entrada = {
      titulo: '',
      resumen: ''
    }
  }

  ngOnInit(): void {
  }

  ngOnChanges(changes: SimpleChanges): void {
    console.log("Valores modificados: ", changes);
  }

}
```

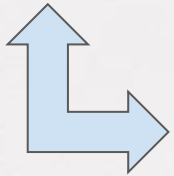
ngOnChanges()

Similar al anterior salvo que ahora añadiremos el método *ngOnChanges()* para detectar la modificación de datos y realizar alguna tarea o comprobación

```html
<div id="contenedor">
  <h3>Listado de entradas al blog:</h3>
  <app-entrada
    *ngFor="let entrada of listadoEntradas"
    [entrada]="entrada" (onMostrarTitulo)="mostrarTitulo($event)"></
app-entrada>
</div>
```

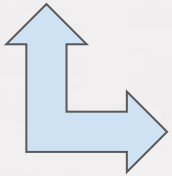
@Output()

Enviaremos eventos desde el componente Hijo al Padre y este ejecutará un método en su componente



```typescript
export class EntradaComponent implements OnInit {
  // Atributos
  @Input()
  public entrada: Entrada;
  @Output()
  public onMostrarTitulo: EventEmitter<string>;

  constructor() {
    this.entrada = {
      titulo: '',
      resumen: ''
    };

    this.onMostrarTitulo = new EventEmitter<string>();
  }

  ngOnInit(): void {
  }

  public lanzarEvent(): void {
    this.onMostrarTitulo.emit(this.entrada.titulo);
  }

}
```

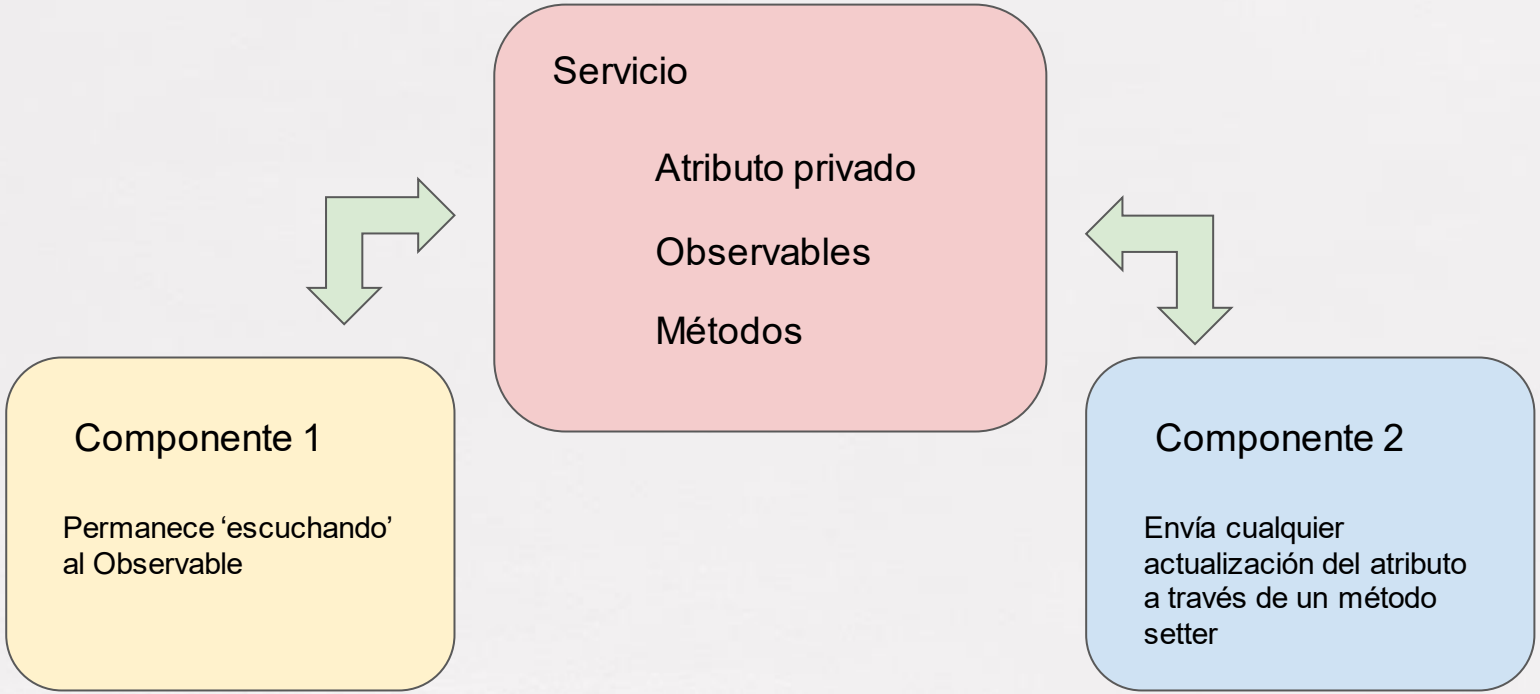
```html
<div id="contenedor">
  <h3>Listado de entradas al blog:</h3>
  <app-entrada #appEntrada [entrada]="entrada" ></app-entrada>
  <button (click)="appEntrada.mostrarTitulo()">Mostrar Titulo</button>
  <button (click)="appEntrada.mostrarResumen()">Mostrar Resumen</button>
</div>
```



A través de la variable #ref

Inicializaremos una variable #ref en la etiqueta del componente hijo y así podremos acceder a los atributos y métodos de este

```typescript
export class EntradaComponent implements OnInit {
  // Atributos
  @Input()
  public entrada: Entrada;

  constructor() {
    this.entrada = {
      titulo: '',
      resumen: ''
    };
  }

  ngOnInit(): void {
  }

  public mostrarTitulo(): void {
    alert(`Titulo: ${ this.entrada.titulo }`);
  }

  public mostrarResumen(): void {
    alert(`Resumen: ${ this.entrada.resumen }`);
  }

}
```



A través de un servicio

Toda la comunicación entre ambos componentes se realiza a través de un servicio intermedio

A través de un servicio

```
export class DataService {
  // Atributos
  private titulo: Subject<string> = new Subject<string>();

  constructor() { }

  // Observables
  public titulo$: Observable<string> = this.titulo.asObservable();

  // Métodos
  public setTitulo(titulo: string): void {
    this.titulo.next(titulo);
  }
}
```