iesperemariaorts

Desarrollo web en entorno cliente

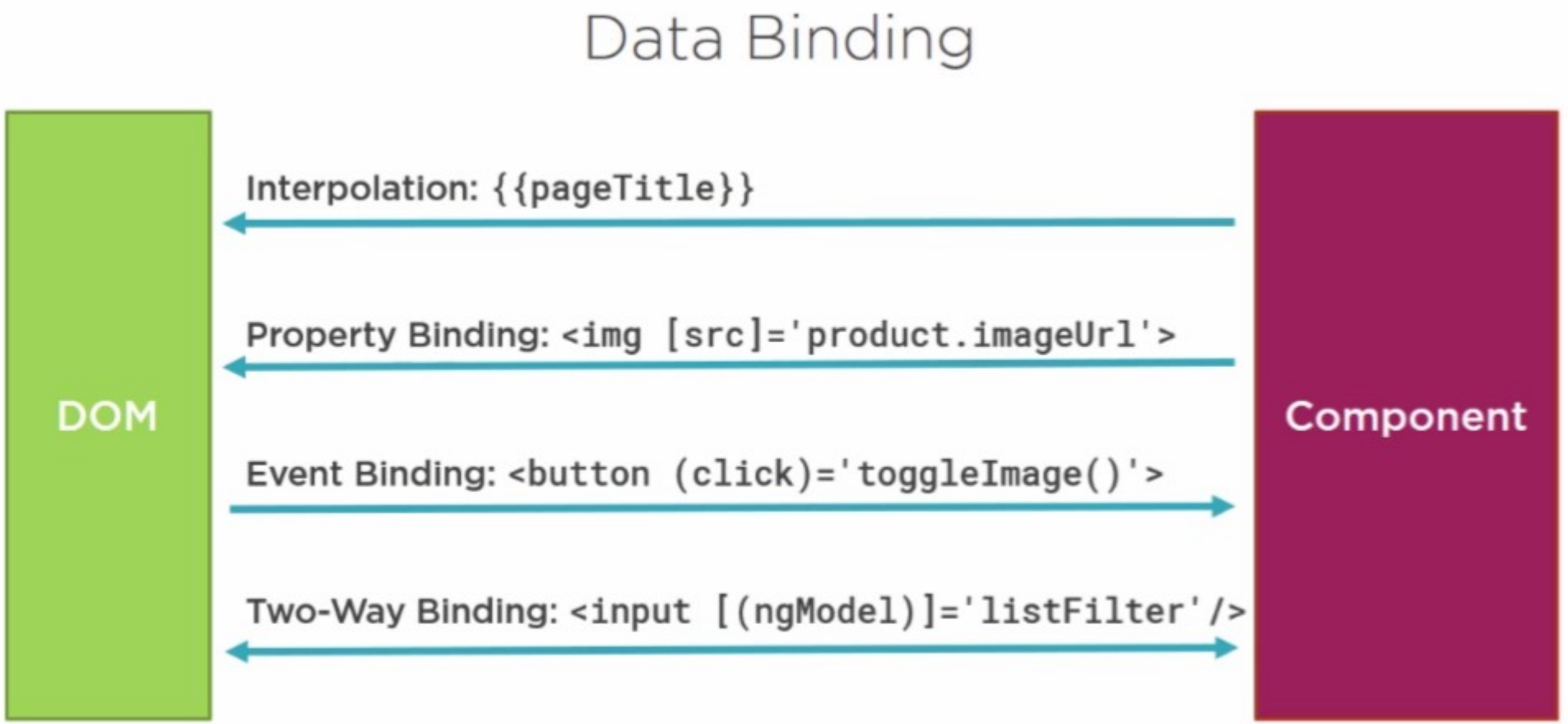
Tema 04

Angular

iesperemariaorts

Angular – Data Binding

- Permite definir la comunicación entre un componente y el DOM ➡ aplicaciones interactivas



iesperemariaorts

Angular – Data Binding - Interpolación

- Se utiliza para mostrar en la vista las propiedades de un componente: {{propiedad}}
- Es la forma más sencilla

src > app > app.component.ts > ...

```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  Libro = {"titulo": "Hamlet", "autor": "William Shakespeare", '
}

```

src > app > app.component.html >

```html
<div class="container-fluid">
  <div class="card" style="width: 20rem;">
    <img class="card-img-top img-fluid" [src]="Libro.imagen" height="122" alt
    <div class="card-block">
      <h4 class="card-title">{{Libro.titulo}}</h4>
      <h6 class="card-subtitle mb-2 text-muted">{{Libro.autor}}</h6>
    </div>
```

iesperemariaorts

Angular – Data Binding – Property Binding

- Utiliza el mismo concepto que la **interpolación** pero aplicado a los atributos de elementos HTML.
- Vincula una propiedad del componente actual a un atributo HTML.

src > app > app.component.ts > ...

```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  Libro = {"titulo": "Hamlet", "autor": "William Shakespeare",
  "precio": 21.30, "stock": 5, "cantidad": 0, "imagen": "assets/old-books.jpg"};
}
```

src > app > app.component.html >

```html
<div class="container-fluid">
  <div class="card" style="width: 20rem;">
    <img class="card-img-top img-fluid" [src]="Libro.imagen" height="122" alt
    <div class="card-block">
      <h4 class="card-title">{{Libro.titulo}}</h4>
      <h6 class="card-subtitle mb-2 text-muted">{{Libro.autor}}</h6>
    </div>
    <ul class="list-group list-group-flush">
```

iesperemariaorts

Angular – Data Binding – Class Binding

- También podemos vincular estilos CSS.

src > app > app.component.css > ...

```
1
2   .aviso{
3       color: red;
4   }
```

src > app > app.component.html > ...

```
1   <div class="container-fluid">
2     <div class="card" style="width: 20rem;">
3       <img class="card-img-top img-fluid" [src]="Libro.imagen" height="122" alt="Imagen no encontrada">
4       <div class="card-block">
5         <h4 class="card-title">{{Libro.titulo}}</h4>
6         <h6 class="card-subtitle mb-2 text-muted">{{Libro.autor}}</h6>
7       </div>
8       <ul class="list-group list-group-flush">
9         <li class="list-group-item" [class.aviso]="Libro.stock==0">Unidades disponibles: {{Libro.stock}}</li
10        <li class="list-group-item">Precio: {{Libro.precio | currency:'EUR':true}}</li>
11      </ul>
12    </div>
13  </div>
```

iesperemariaorts

Angular – Data Binding – Event Binding

- Sería lo "opuesto" de la vinculación de atributos.
- Vincularíamos desde la plantilla al componente.
- Capturamos eventos del DOM y lo transmitimos al componente para realizar acciones (click, keypress...)

```
    </ul>
    <div class="text-center">
      <button [disabled]="Libro.cantidad<=0" (click)="downCantidad(Libro)">-</button>
      {{Libro.cantidad}}
      <button [disabled]="Libro.cantidad>=Libro.stock" (click)="upCantidad(Libro)">+</button>
    </div>
  </div>
</div>
```

```
downCantidad(libro){
  if (libro.cantidad > 0 ) libro.cantidad--;
}
upCantidad(libro){
  if (libro.cantidad < libro.stock ) libro.cantidad++;
}
getCoord(event) {  console.log(event.clientX + ", " + event.clientY); }
```

iesperemariaorts

Angular – Data Binding – Two-Way Binding (ngModel)

- Comunicación bidireccional entre componente y plantilla
- Directiva [(ngModel)]: se vincula, normalmente, con elementos <input>. Se vincula el valor del campo a una propiedad del componente. Debemos importar FormsModule.

```
<div class="text-center">
  <button [disabled]="Libro.cantidad<=0" (click)="downCantidad(Libro)">-</button>
  <input type="text" [(ngModel)]="Libro.cantidad" size="2">
  <button [disabled]="Libro.cantidad>=Libro.stock" (click)="upCantidad(Libro)">+</button>
</div>
```

```
export class AppComponent {
  Libro = {"titulo": "Hamlet", "autor": "William Shakespeare",
  "precio": 21.30, "stock": 5, "cantidad": 0, "imagen": "assets/old-books.jpg"};
```

iesperemariaorts

Angular – Filtros (Pipes)

- Se combinan con la interpolación para transformar la información antes de mostrarla.
- Angular provee algunos filtros personalizados.
- Estos filtros van detrás del valor a transformar, concatenados con el carácter '|'.
- Algunos filtros admiten parámetros que pasaremos separados por el caracter ':'

```
export class DatePipeComponent implements OnInit {
  fecha: Date = new Date(1982, 4, 7, 12, 40, 11); //7 de Mayo del 1982 12:40:11
```

```
    <td ngNonBindable>{{ fecha | date | uppercase}}</td>
    <td>{{ fecha | date | uppercase}}</td>
  </tr>
  <tr>
    <td ngNonBindable>{{ fecha | date:'medium' | lowercase}}</td>
    <td>{{ fecha | date:'medium' | lowercase}}</td>
  </tr>
  <tr>
```

iesperemariaorts

Angular – Filtros (Pipes) Personalizados

- Podemos crear filtros personalizados
  - ng g pipe ruta/nombre-filtro
- Método transform es el encargado de aplicar el filtro.

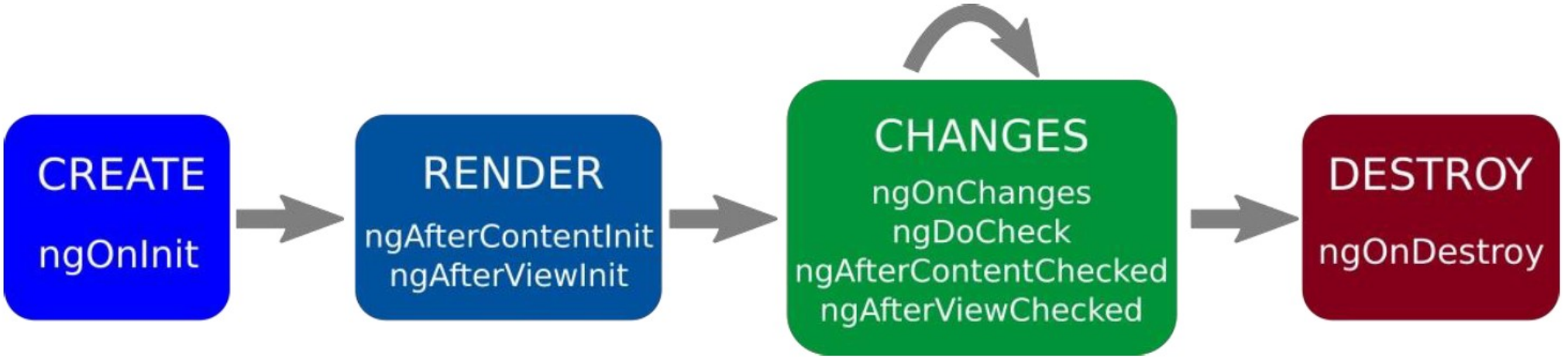
```
<p><b>Pipes personalizados: divisonEnteraPipe</b></p>
<div>Dividendo: <input [(ngModel)]="dividendo"></div>
<div>Divisor: <input [(ngModel)]="divisor"></div>
<p>  Resultado: {{dividendo | divisionEntera: divisor}}</p>
```

```
@Pipe({
  name: 'divisionEntera'
})
export class DivisionEnteraPipe implements PipeTransform {

  transform(value: number, divisor: string): string {
    let div = parseFloat(divisor);
    let cociente = Math.floor(value/div);
    let resto = value % div;

    return "Cociente: " + cociente + " # Resto: " + resto;
  }
```

iesperemariaorts

Angular – Ciclo vida de componentes

- Los componentes pasan por varias etapas desde que se crean hasta que se destruyen.



Angular – Ciclo vida de componentes

```
export class Compo1Component implements OnInit {
  @Input() entradaHijo: string = "";
  contador: number = 0;
  constructor() { }
  ngOnInit() { this.mostrar("pasa por ngOnInit"); }
  ngOnChanges(cambios: SimpleChanges) {
    for (let propiedad in cambios) {
      let cambio = cambios[propiedad];
      let actual = JSON.stringify(cambio.currentValue);
      let anterior = JSON.stringify(cambio.previousValue);
      this.mostrar("Pasa por ngOnChanges. Propiedad (" + propiedad + ") valor actual (" +
    }
  }
  ngDoCheck() { this.mostrar("pasa por ngDoCheck"); }
  ngAfterContentInit() { this.mostrar("pasa por ngAfterContentInit"); }
  ngAfterContentChecked() { this.mostrar("pasa por ngAfterContentChecked"); }
  ngAfterViewInit() { this.mostrar("pasa por ngAfterViewInit"); }
  ngAfterViewChecked() { this.mostrar("pasa por ngAfterViewChecked"); }
  public mostrar(texto: string) {
    this.contador += 1;
    console.log(this.contador + " - " + texto);
```