## 1. PROJENİN OLUŞTURULMASI

**ng new example** ile example adında yeni bir proje oluşturalım.

Proje klasörüne geçiş yapmak için code terminalinde **cd example** dedikten sonra **code .** diyerek yeni bir ekranda ilgili projeyi vs code ile açarız.

Proje yapısına hakimiz ancak burada hızlıca bi tekrar yazıyla üzerinden geçelim:

- Package.json dosyasında npm aracılığı ile indirilen paketler ve npm scriptleri yer alıyor. Bu indirilen module'ler de node modules klasörü içerisinde durur. Projeyi taşırken, node_modules'ü taşımaya gerek yok zaten node_modules'ün içeriğindekileri package.json belirliyor bu yüzden package.json varsa npm install dediğimizde zaten gerekli node_module indirilmiş olur.
- Projemiz ise src altındaki app klasöründe yer alır.
- App klasöüründe bir app.module'ümüz var burada angular'dan gelen module'ler ile bizim oluşturduğumuz component'ler paketlenir. Burada ayrıca başlangıç component'i de tanımlanır default olarak başlangıç component'i AppComponent'tir.
- Bu module dışarıya açılır ve bu module'ü kullanan program ilk olarak AppComponent'i çalıştırır.
- Main.ts içerisinde ise projenin browser üzerinde çalışacağı söylenmiş ve başlangıçta hangi module'ün çalıştırılacağı da burada AppModule olarak belirtilmiş.

Şimdi daha önce yaptığımız gibi projeye bootstrap kütüphanesi indirelim:

- **npm install bootstrap** dedikten sonra
- angular.json altında styles arrayinin içine aşağıdaki 2. Satırı ekliyoruz:

```
              "styles": [
                "src/styles.css",
                "node_modules/bootstrap/dist/css/bootstrap.min.css"
              ],
```

- Artık bootstrap css'ini kullanabiliyoruz, jquery.js ve bootstrap.js gerekli olursa onları da bir alttaki scripts arrayine ekleriz. Bunu zaten daha önce gördük.

## 2. DATA MODELİNİN EKLENMESİ

Şimdi projeye bir product modeli ekleyelim. Bunun için app klasörü altına product.model.ts isminde bir file ekliyoruz. Bu klasör içinde yapılan bir Product class'ı tanımlamak, özelliklerini vermek:

product.model.ts

```
export class Product {
    constructor(
        public id?: number,
        public name?: string,
        public description?: string,
        public imageUrl?: string,
        public price?: number
    ){}
}
```

Şimdi ise bu Product class'ından objeler üreteceğiz ve bir datasource class'ı oluşturacağız:

Yeni bir datasource.model.ts file'ı altında:

```
import { Product } from './product.model';

export class SimpleDataSource{
    private products: Product[];
    constructor(){
        this.products = new Array<Product>(
            new Product(1, "Samsung S5", "İyİ telefon", '1.jpg', 1000),
            new Product(2, "Samsung S6", "İyİ telefon", '2.jpg', 2000),
            new Product(3, "Samsung S7", "İyİ telefon", '3.jpg', 3000),
            new Product(4, "Samsung S8", "İyİ telefon", '4.jpg', 4000),
            new Product(5, "Samsung S9", "İyİ telefon", '5.jpg', 5000),
            new Product(6, "Samsung S10", "İyİ telefon", '6.jpg',6000)
        );
    }

    getProducts(): Product[] {
        return this.products;
    }
}
```

Göründüğü gibi SimpleDataSource class'ından bir obje üretilince, products attribute'u yukarıdaki gibi 6 tane product ile dolacak ve getProducts methodu ile bu attribute'a ulaşılabilecek.

Şimdi bu bölümün sonu olarak bir repository.model.ts dosyası oluşturulacak ve içindeki ProductRepository class'ı ile bu datasource'dan veriler çekilecek, bu işlem bence DataSource class'ında da yapılabilirdi ama, böyle yapılmamış.

Sonuçta farketmez aşağıdaki class'dan bir obje üretildiğinde artık yukarıda tanımlanan 6 product products içine alınmış olacak .

repository.model.ts

```
import { SimpleDataSource } from './datasource.model';
import { Product } from './product.model';

export class ProductRepository {
    private dataSource: SimpleDataSource;
    private products: Product[];

    constructor(){
        this.dataSource =  new SimpleDataSource();
        this.products = new Array<Product>();
        this.dataSource.getProducts().forEach(p=>this.products.push(p));
    }

    getProducts(): Product[]{
        return this.products;
    }

    getProductsById(id: number): Product {
        return this.products.find(p=>p.id==id);
    }
}
```

Repository üzerinde getProduct veya getProductsById gibi methodlar tanımlandığına dikkat et daha sonra eleman ekleme, silme, update etme gibi işlemleri de biz buraya methodlar ekleyerek yapacağız.

## 3. PRODUCT COMPONENT'ININ EKLENMESİ

Bu kısımda yapacağımız şe app component'i silmek bunun yerine bir product component'i oluşturmak ve bu comonent'in de projenin giriş component'i olmasını sağlamak.

- Öncelikle app.component ile alakası 4 dosyayı siliyoruz: .ts, .css, .html ve .spec olanları.

Bu dosyayı sildik bunun yerine şimdi buna benzer kendimiz bir component oluşturacağız.

Bu arada AppComponent'i sildiğimiz için app.module.ts içerisinde declare edilen ve başlangıç component'i olarak belirtilen kısımları da silmemiz lazım:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Şimdi projeye bir product component'i ekleyelim:

- **ng g c product**

Bunu oluşturunca direkt olarak 4 farklı component dosyası product klasörü altında oluşacak.

Product.component.ts dosyasına bakarsak:

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

@Component({
  selector: 'app',
  templateUrl: './product.component.html',
  styleUrls: ['./product.component.css']
})
export class ProductComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

Şimdi bu component'in içini biraz dolduralım ve burada bir ProductRepository objesi tanımlayalım böylece repository üzerinden product'lara ve gerekli repository methodlarına ulaşabiliriz. Product.component.ts

```
import { Component, OnInit } from '@angular/core';
import { ProductRepository } from '../repository.model';

@Component({
  selector: 'app',
  templateUrl: './product.component.html',
  styleUrls: ['./product.component.css']
})
export class ProductComponent implements OnInit {

  model: ProductRepository = new ProductRepository();

  constructor() { }

  ngOnInit() {
  }

}
```

Artık bu ProductRepository modeline view içinden yani template içinden ulaşılabilir. product.component.html

```
<div class="bg-primary text-white m-2 p-2">
    There are {{model.getProducts().length}} products
    in the repository.
</div>
```

Göründüğü gibi string interpolation ile model'in product sayısına ulaştık. Ayrıca app.module.ts içinde yeni Component'i declare ediyoruz ve bootstrap component olarak belirtiyoruz:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { ProductComponent } from './product/product.component';

@NgModule({
  declarations: [
    ProductComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [ProductComponent]
})
export class AppModule { }
```

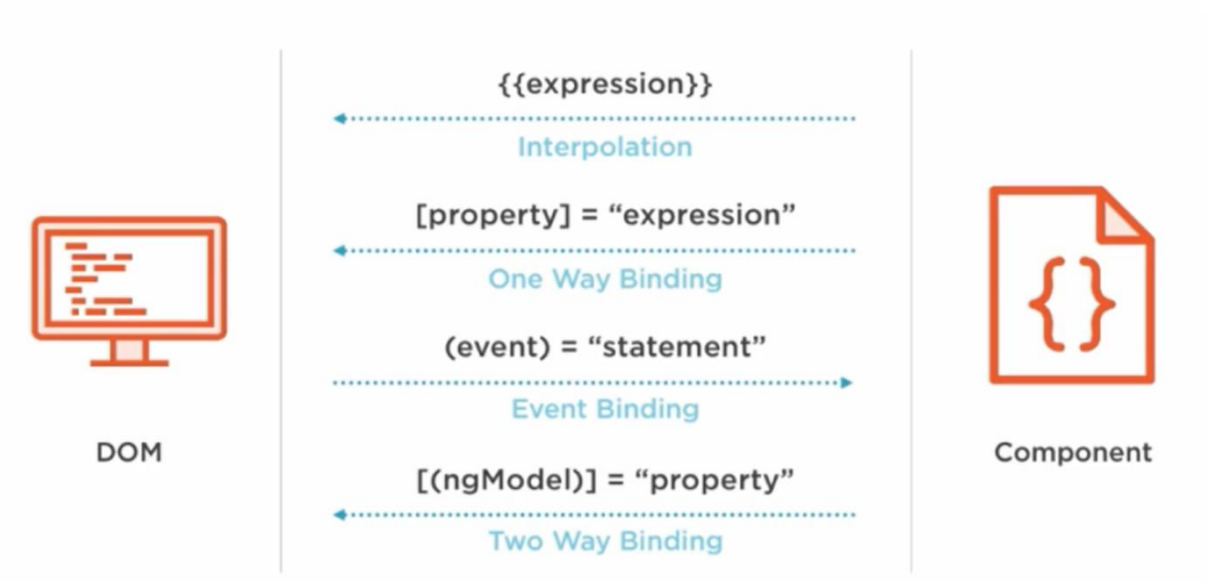
Son olarak index.html içerisinde product component'i app selector'ı ile çağırıyoruz.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Example</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app></app>
</body>
</html>
```

## 4. PROPERTY BINDING

Component.ts içerisinde oluşturulan bir içeriği html sayfasında göstermek istiyoruz, bunun için farklı yöntemlerden söz edilebilir.



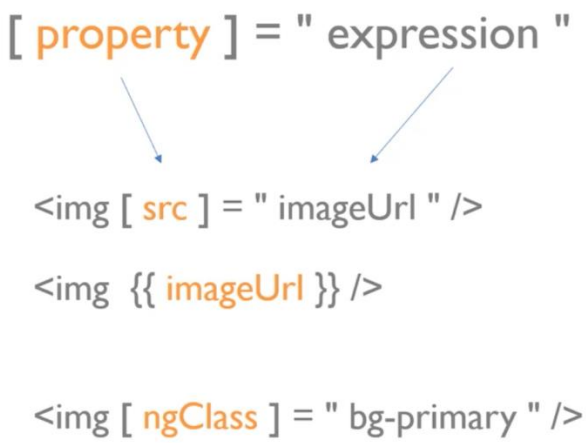

- Örneğin component içinde tanımlanan bir değişkene template içerisinde string interpolation "{{}}" ile ulaşabiliriz.
- İkinci alternatif yukarıda da görülen property bindingdir.
- Diğer bir binding yöntemi ise event binding. Burada bilginin ters yönlü aktığına dikkat et. Burada bilgi template'ten bir event ile bir method içerisinde component'e gelir.
- Son olarak da zaten bildiğimiz ngModel ile two way binding yapmak var, böylece component içerisinde tanımlanan içerik, view'den ulaşılabilir olduğu gibi view içerisinden yapılan değişiklik de anlık olarak component içerisindeki değeri etkiler.

Bunların 3'ünü zaten biliyorum, property binding'i de bir html elemanına class eklemek için kullandık. Eğer component içinde tanımlanan değişkenler'in oluşturduğu bir expression true ise html elemanına bir class eklemiştik.

<li> class= "xyz" **[class.active] = "movie === selectedMovie"** </li>

Şeklinde kullanmıştık.

Tüm binding yöntemlerinin detayına girilecek ancak bu kısımda property binding'e yoğunlaşalım.




- Burada expression component içindeki değişkenlerden oluşabildiği gibi, methodlardan da oluşabilir.
- Örneğin yukarıdaki [src] = "imageUrl" kullanımı ile img elemanın src attribute'una component içerisinden alınan imageUrl'nin değeri atanıyor.
- Burada interpolation da kullanılabilirdi.
- Ancak bazı direktifler de bu şekilde kullanılır. Yukarıdaki ngClass örneğine bir sonraki başlıkta değinilecek.

PROPERTY BINDING'I KULLANMAK İÇİN [] İÇİNE YAZILAN PROPERTY'NIN İLGİLİ HTML ELEMANINA AİT OLAN BİR PROPERTY OLMASI LAZIM!

Şimdi  bir örnekle  string interpolationi ve property binding'i daha iyi anlayalım:

Product.component.ts içerisinde model isminde bir productrepository objesi oluşturmuştuk, bu obje içerisinden products listesine ulaşabildiğimiz gibi, getProductsById methodu ile 1 numaralı id'ye sahip olan ürünü de elde edebiliriz:

```
export class ProductComponent implements OnInit {

  model: ProductRepository = new ProductRepository();
  product: Product = this.model.getProductsById(1);
  constructor() { }

  ngOnInit() {
  }

}
```

Artık bu product, product template'i içerisinden kullanılabilir. Yani product.component.html içerisinden:

```
<div>
  <h2>{{product.name}}</h2>
  <h2 [textContent]="product.name"></h2>

  <img src = "assets/img/{{product.imageUrl}}">
  <img [src] = "'assets/img/'+product.imageUrl ">
</div>
```

- String interpolation ile component içerisindeki product'a zaten direkt ulaşabiliyoruz.
- Benzer şekilde property binding ile html elemanın DOM property'lerini kullanarak, component içerisindeki değerleri property'lere atayabiliyoruz.
- Burada textContent zaten h2 elemanın sahip olduğu bir attribute'tur. Bu yüzden kullanılabiliyor. Aynı js ile getElementById diyip elemanı alıp .textContent = "xyz" demek gibi bir şey bu.
- Benzer şekilde img'in src etiketine bir değer atamak için string interpolation'ı kullanabileceğimiz gibi, property binding ile de bu işlemi yapabiliriz.
- Property binding yaparken " " içerisinde  ' ' kullanarak bir string ile component değişkenini bağlayabildiğimize dikkat et.

## 5. CLASS BINDING

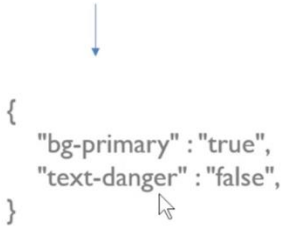
Önceki kısımda da bahsedilen property binding başlığı altında class binding'den bahsedilebilir.

Class binding için 3 farklı yöntemden söz etmek mümkün:

```
<div [class] = "expression"></div>

<div [class.active] = "expression"></div>

<div [ngClass] = "object"></div>
```



```
{
   "bg-primary" : "true",
   "text-danger" : "false",
}
```

- İlk yöntem ile bir expression'ın sonucuna bağlı olarak div elemanına bir class eklenebilir. Burada dikkat edilmesi gereken husus şu, eğer div için halihazırda bazı class'lar tanımlanmışsa ayrıca expression'dan bir class atarsak, var olan class'lar silinecektir.
- İkinci yöntem ise halihazırda var olan bir class'a mesela active class'ını eklemek istiyorsam bunu kullanırım.
- Son yöntem ise daha genel bir yöntem, ngClass'a gelecek değerleri bir object olarak getirebiliyoruz, yukarıda göründüğü gibi ngClass ile örneğin ürünün fiyat bilgisi bir şeyin üstündeyse ürüne bg-primary class'ı eklensin, eğer ürün son hafta eklendiyse text-danger class'ı eklensin gibi şeyleri bir json objesi ve ngClass kullanarak yapabiliyorum.

Şimdi pratik yapmak için bu yöntemleri projemizde inceleyelim. Önce ilk yönteme bakalım:

Öncelikle product.component.ts içerisnde bir getClasses methodu tanımlıyoruz, bu methoda bir id veriliyor ve sonucunda ilgili id'ye sahip olan product.price eğer <= 1000 ise "bg-info" + " m-2 p-2" return et yok eğer product.price >1000 ise "bg-secondary" + " m-2 p-2" return et:

```
export class ProductComponent implements OnInit {

  model: ProductRepository = new ProductRepository();
  product: Product = this.model.getProductsById(1);
  constructor() { }

  ngOnInit() {
  }

  getClasses(id: number): string {
      let product = this.model.getProductsById(id);
      return (product.price <= 1000 ? "bg-info":"bg-secondary") + " m-2 p-2"
  }

}
```

Şimdi bu getClasses metodunu product view içerisinde kullanalım.
Product.component.html

```
<div [class] = "getClasses(1)" >
  İlk ürün: {{model.getProductsById(1).name}}
</div>

<div [class] = "getClasses(2)" >
  İkinc ürün: {{model.getProductsById(2).name}}
</div>
```

- Burada ne yapıldı, ilgili div'e component içindeki bir değişkeni kullanarak class'lar atandı, bu class'ları atamak için arasında boşluk bırakılmış string'ler döndürüldü.
- Yani mesela id=1 ürünü için getClasses(1) methodu bize "bg-info m-2 p-2" değerini döndürdü ve bu değer de div'in class'ı olarak atandı.

Unutma eğer burada view içerisinde div halihazırda bazı class'lara sahip olsaydı, class binding yüzünden bu class'lar işlevsiz kalacaktı, yeni class'lar üzerine eklenmek yerine baştan yazılacaktı.

Devam edelim şimdi ikinci yönteme bakacağız. diyelim ki ben view'e bir buton ekliyorum:

```
<button class="btn btn-primary">Submit</button>
```

Eğer bu butona bir disabled class'ı eklenirse, buton soluk renk olur ve tıklanması engellenmiş olur.

```
<button class="btn btn-primary disabled">Submit</button>
```

Ben butonun disable class'ına sahip olup olmamasını duruma göre istiyor olabilirim. Öncelikle product.component.ts içerisinde bir isDisabled değişkeni tanımlayalım, true olsun:

```
export class ProductComponent implements OnInit {

  model: ProductRepository = new ProductRepository();
  product: Product = this.model.getProductsById(1);
  isDisabled : 12boolean = true;
```

Şimdi ben butona view içerisinde disabled özelliği vermek için ilk yöntemi kullanırsam problem yaşarım:

```
<button class="btn btn-primary" [class]="isDisabled" >Submit</button>
```

Dersek butonun class'ına true değeri gelir, bu hata ayrıca diğer class'lar da silinir.

Bunun için aşağıdaki şekilde ikinci class binding yöntemini kullanıyoruz:

```
<button class="btn btn-primary" [class.disabled]="isDisabled">Submit</button>
```

Burada eğer, isDisabled değişkeni true ise butona disabled class'ı eklenecek yok false ise buton class'ı olduğu gibi kalacak

Son olarak üçüncü class binding yöntemi olan ngClass yapısını kullanalım:

Öncelikle product.component.ts içerisinde getClassMap adında yeni bir method tanımlıyoruz.

```
export class ProductComponent implements OnInit {

  model: ProductRepository = new ProductRepository();
  product: Product = this.model.getProductsById(1);
  isDisabled : boolean = true;
  ...
  getClasses(id: number): string {
      let product = this.model.getProductsById(id);
      return (product.price <= 1000 ? "bg-info":"bg-secondary") + " m-2 p-
2 text-white"
  }

  getClassMap(id: number): Object {
    let product = this.model.getProductsById(id);
    return {
        "bg-info": product.price <= 1000,
        "bg-secondary": product.price > 1000,
        "text-center text-white": product.name == "Samsung S6"
    }
  }
}
```

Görüldüğü üzere bu method bir id alıp bu id'ya karşılık, bir object return ediyor. Bu object içerisinde ise farklı class isimleri ve buna karşılık true ya da false olacak boolean statement'lar yer alıyor.

Sonuçta bu object view'e return edilecek ve ngClass yapısı ile birlikte kullanılınca, object içinde hangi class'ın karşılığı true ise elemana o class eklenecek. Product.component.html

```
<div class= "m-2 p-2"  [ngClass]="getClassMap(1)">
  İlk ürün: {{model.getProductsById(1).name}}
</div>

<div class= "m-2 p-2" [ngClass]="getClassMap(2)">
  İkinci ürün: {{model.getProductsById(2).name}}
</div>
```

Böylelikle ürünlere eklenecek class'lar component içerisinde bazı kriterler ile belirlendi ve sonuçlar view'e aktarıldı.

## 6. STYLE BINDING

Class'ların nasıl atanacağını gördük, şimdi CSS kodlarını nasıl etiketlerle ilişkilendireceğiz onu anlayalım.

Class binding'e baya benziyor aslında.

Diyelim ki view içerisinde aşağıdaki div'in içine bir span tanımlandı ve bu span'e component'ten inline css geçirmek istiyoruz:

```
<div class= "m-2 p-2"  [ngClass]="getClassMap(1)">
  <span>İlk ürün: </span> {{model.getProductsById(1).name}}
</div>
```

Eğer tek bir css özelliği'ni componentten view'e geçirmek istiyorsak:

```
<div class= "m-2 p-2"  [ngClass]="getClassMap(1)">
  <span [style.color] = "color" >İlk ürün: </span> {{model.getProductsById(1).
name}}
</div>
```

- Yukarıdaki şekilde, aynı elemana class atıyormuşuz gibi yapıyoruz. Burada "color" daki color bilgisinin component içinden geldiğini unutma:

```
color: string = "red";
```

Buna benzer şekilde eğer component içerisinde bir fontSize değişkenini

```
fontSize: string = "25px";
```

Şeklinde tanımlarsak, view içerisinde inline olarak span'e ilgili font size'ı atayabiliriz:

```
<div class= "m-2 p-2"  [ngClass]="getClassMap(1)">
  <span [style.fontSize] = "fontSize" >İlk ürün: </span> {{model.getProductsBy
Id(1).name}}
</div>
```

Elbette component içerisinde bu style değerlerini koşullara göre belirleyebiliriz, örneğin bir ürünün fiyatı 2000'den fazlaysa color özelliğine kırmızı değilse yeşil ekleyebiliriz gibi.

Dikkat ettiysen: span'in stiline aynı anda hem color hem fontsize'ı bu şekilde veremeyiz.

## Eğer birden fazla css özelliği'ni view'e geçirmek istiyorsak:

Burada da yapılacak işlem ngClass kullanmaya çok benziyor, component içerisinde aşağıdaki gibi bir getStyles methodu tanımlarız:

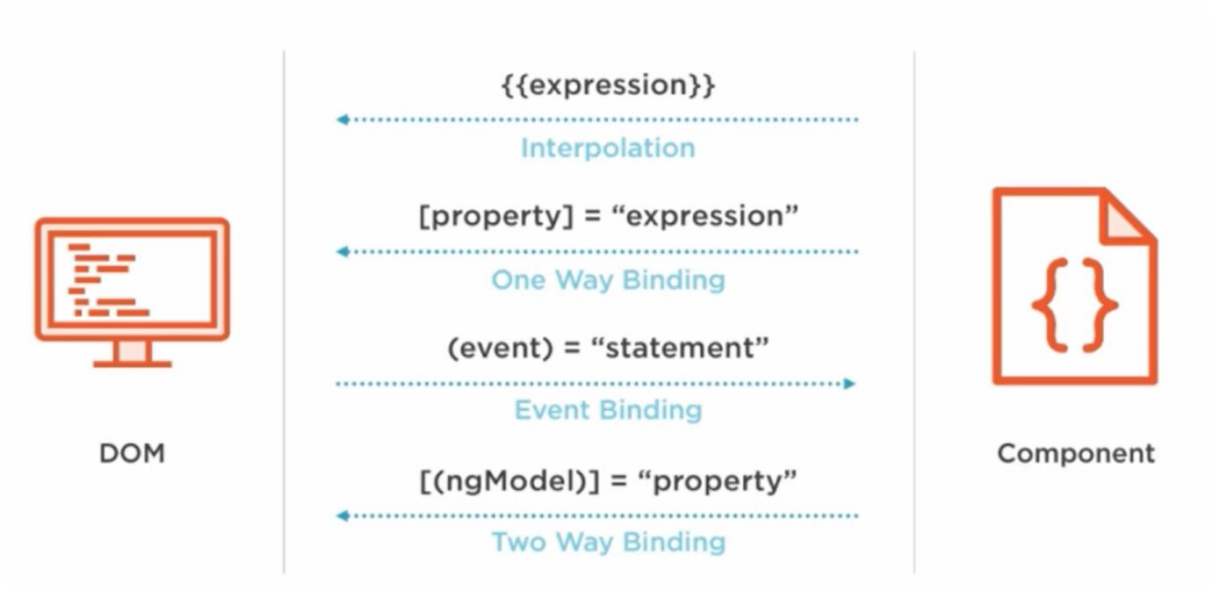
```
  getStyles(id:number){
    let product =  this.model.getProductsById(id);
    return {
        fontSize: "25px";
        color: product.price <= 1000 ? "green" : "red";
    }
  }
```

Burada return'ün bir objec olduğuna dikkat et,  ngClass kullanımına benzer şekilde, inline css ataması için bir object döndürüyoruz.

```
<div class= "m-2 p-2" [ngClass]="getClassMap(2)">
  <span [ngStyle]="getStyles(2)" >İkinci ürün: </span> {{model.getProductsById
(2).name}}
</div>
```

Yukarıdaki gibi ngStyle directive'i ile ilgili CSS bilgileri view içine geçiriliyor. Buradaki yapı ngClass'a benzese de tam aynı değil ngClass'da return edilen object class: true/false şekindeyken buradaki yapı property:value şeklinde.

BÖYLELİKLE ASLINDA ÖNCEKİ KISIMLARDA BAHSEDİLEN BINDING YONTEMLERINDEN IKISINI TAMAMLAMIŞ OLDUK:



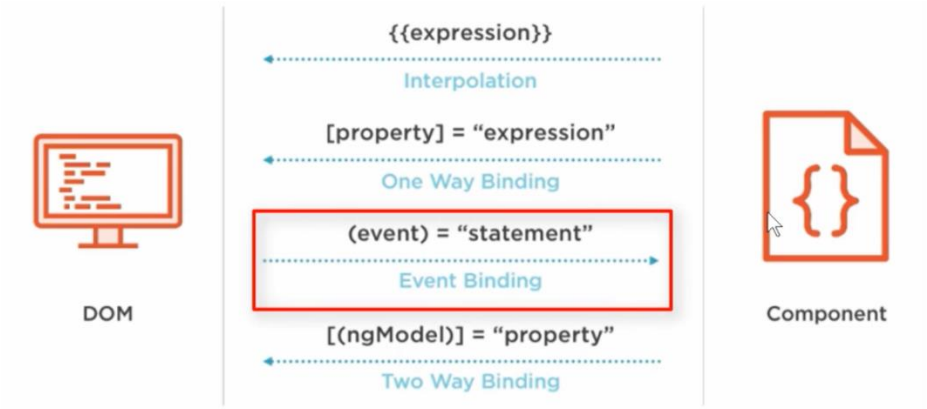

String interpolation zaten intuitive bunda sıkıntı yok, property binding’in de birsürü çeşidi olduğunu anlamış olduk hem html elemanlarının dom attribute’ları kullanılarak attribute atamak için kullanılabiliyor. Ya da 3 farklı şekilde class binding için kullanılabiliyor. Ya da benzer şekilde style binding için bu yapı kullanılabiliyordu.

İlerleyen kısımlarda ise kalan binding çeşitlerine bakalım. Event binding ve ngModel kullanımlarına bakarak binding konusunu iyice anlamaya çalışalım.

## 7. EVENT BINDING

Şimdi de event binding'e bakalım. Bu binding'in diğer ikisinden farkı şu: Event binding ile Component'ten DOM'a değil de DOM'dan Component'e bir akış vardır.

Html sayfasından alınan değerler, component'e event binding ile gönderilebilir.




Şimdi product.component.html içerisindeki eski içerikleri silelim, sadece bir buton yerleştirelim ve butona da bir click event'i tanımlayalım.

```
<button (click)="onSubmit()">Submit</button>
```

Görüldüğü gibi bu botna tıklandığında component içerisindeki onSubmit() methodu çağırılacak.

Elbette component içerisinde bu methodu da doldurmalıyız, örneğin console'a yazı yazdıracak bir onSubmit methodu tanımlanabilirdi:

```
export class ProductComponent implements OnInit {

  model: ProductRepository = new ProductRepository();

  onSubmit(){
    console.log('Butona tıklandı!')
  }

}
```

Şimdi ise template üzerinden butona tıklandığında, event ismindeki bir parametreyi component'e gönderelim:

```
<button (click)="onSubmit($event)">Submit</button>
```

Anladığım kadarıyla bu event parametresi özel bir parametre, bu bilgi tarayıcıdan alınıyor, hangi event'in geldiğini, butonun konumunu, ismini vesaire birsürü bilgiyi içinde taşıyor ve component'e götürüyor.

```
export class ProductComponent implements OnInit {

  model: ProductRepository = new ProductRepository();

  onSubmit($event){
    console.log('Butona tıklandı!');
    console.log($event);
  }

}
```

Sonuçta console'da şöyle bir şey görüyoruz:

*MouseEvent {isTrusted: true, screenX: 68, screenY: 120, clientX: 54, clientY: 14, …}*

Eğer sayfada iki farklı buton olsaydı:

```
<button name="btn1" (click)="onSubmit($event)">Submit</button>
<button name="btn2" (click)="onSubmit($event)">Submit</button>
```

$event'i parametre göndererek, hangi butona tıklandığını component içerisinde yakalanan $event'e bakarak vesaire anlayabilirdik.

Eğer ki bu butonları bir div içine alsaydık ve div'e de bir click event'i tanımlasaydık:

```
<div (click)="onDivClicked()">
  <button name="btn1" (click)="onSubmit($event)">Submit</button>
  <button name="btn2" (click)="onSubmit($event)">Submit</button>
</div>
```

Buna karşılık component'in içine de bu onDivClicked()'i tanımlarsak:

```
export class ProductComponent implements OnInit {

  model: ProductRepository = new ProductRepository();
  onSubmit($event){
    console.log('Butona tıklandı!');
    console.log($event);
  }

  onDivClicked(){
    console.log('Dive tıklandı');
  }
}
```

Artık sayfadaki hangi butona tıklarsak tıklayalım, burada buton div'in içinde olduğu için dive de tıklanmış olacak ve her seferinde onDivClicked() methodu da çalışacaktır.

Bunu durdurmak istersek, butona tıklandığında çalışan onSubmit() methodunu aşağıdaki gibi modify ederiz:

```
export class ProductComponent implements OnInit {

  model: ProductRepository = new ProductRepository();

  onSubmit($event){
    $event.stopPropagation();
    console.log('Butona tıklandı!');
    console.log($event);
  }

  onDivClicked(){
    console.log('Dive tıklandı');
  }
}
```

Böylece, butona tıklayınca event yukarıya doğru yayılmaz sadece onSubmit çalışır ancak onDivClicked çalışmaz.

## 8. KEYUP EVENT

Click event'ini gördük, şimdi bir başka event olan keyup eventini görelim. KeyUp event'i keyboard'da bir key'e basılınca çalışır.

Öncelikle component.html içerisinde aşağıdaki gibi bir input tanımlayalım ve bu input'a bir keyup event'i ekleyelim, event çalışınca onKeyUp() methodu çalışacak ve içinde browser'dan $event objesini alacak.

```
<input (keyup)="onKeyUp($event)">
```

Input içerisinde herhangi bir key'e basıldığında keyup event'i trigger olur ve onKeyUp methodu çalışır, $event methodunu kullanarak da gerçekleşen event ile ilgili detaylı bilgileri component'e taşıyabiliriz.

Örneğin component içerisinde $event.keyCode ile hangi tuş ile event'in trigger edildiğini anlayabiliriz. Her tuşun bir keyCode'u vardır.

```
export class ProductComponent implements OnInit {

  model: ProductRepository = new ProductRepository();

  constructor() { }

  ngOnInit() {
  }

  onKeyUp($event){
    console.log($event.keyCode)
  }

}
```

Benzer şekilde,

```
<input (keyup.enter)="onKeyUp()">
```

Dersek, keyup event'i sadece enter tuşuna basıldığında trigger edilecektir.

```
  onKeyUp($event){
    console.log("Enter is pressed")
  }
```

Peki bir input içerisine yazılan bir değer, componente nasıl taşınır?

- İlk yol biraz daha karmaşık bir yol:

```
<input (keyup.enter)="onKeyUp($event)">
```

- Diyerek daha önce yaptığımız gibi, enter'a basılınca onKeyUp çalışacak ve event objesini de component'e taşıyacak.
- İşte bu event objesi aslında içerisinde input'un value'sunu da taşıyor buna ulaşmak için component içerisinde:

```
  onKeyUp($event){
    console.log($event.target.value);
  }
```

- Böylece event trigger olunca, $event objesi üzerinden target alınır yani event'in hangi nesne üzerinde gerçekleştiği alınır ki bu input nesnesine karşılık gelir, daha sonra bu nesnenin event trigger edildiğindeki value'su yani içeriği elde edilebilir.

- İkinci yol ise görece daha kolay bir yol
- Inputa # işareti ile bir id verilir ve daha sonra, event trigger olunca çağıralacak method içerisinde bu id'ye sahip elamanın value'su parametre olarak gönderilir:

```
<input #inp1 (keyup.enter)="onKeyUp(inp1.value)">
```

- Daha sonra bu value component içerisinde onKeyUp methodunda yakalanır:

```
  onKeyUp(val){
    console.log(val);
  }
```

## 9. TWO WAY BINDING

Şimdi ise daha önce de ngModel ile kullandığımız two way binding'e değineceğiz.

ngModel directive'i ile two way binding yapabilmek için öncelikle app.module.ts içerisinde FormsModule'ü import etmemiz gerekiyor!

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { ProductComponent } from './product/product.component';
import { FormsModule } from '@angular/forms';

@NgModule({
  declarations: [
    ProductComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [ProductComponent]
})
export class AppModule { }
```

Daha sonra product.component.html içerisine bir input tanımlayalım, ve input'un içeriğini ngModel ile value ismindeki component değişkeni ile bind edelim:

```
<input [(ngModel)]="value" (keyup.enter)="onKeyUp()">
```

Burada ngModel directive'i  ile input'un içeriği component'deki value değeri ile bind edilirken bir de (keyup.enter) event'i ile Enter'a basılınca onKeyUp() methodu çalıştırılıyor.

```
export class ProductComponent implements OnInit {
  …
  value = "initialValue";

  onKeyUp(){
    console.log(this.value);
  }
}
```

Artık input içerisine girilen değer anlık olarak component'in value değeri içerisine atanacak, aynı zamanda component içindeki bir değişim de input içerisinde görünecektir.

Bu işlemi daha önce öğrendiklerimizle ngModel kullanmadan da yapabilirdik, onu da görelim ancak ngModel daha pratik.

Öncelikle component'ten DOM'a property binding ile bir val değişkenini taşıyalım:

```
export class ProductComponent implements OnInit {

  val = "initialValue";

  onKeyUp(){
    console.log(this.val);
  }

}
```

Yukarıdaki gibi component içerisinde val değişkenine bir değer atandı, artık property binding ile input elemanının value attribute'una val değeri atanabilir.

```
<input [value]="val" (keyup.enter)="onKeyUp()">
```

Ancak şuna dikkat et, keyup.enter methodu çalışınca console'da yazdırılacak olan değer direkt component'ten geliyor, yani bizim view içerisinde input'u değiştirmemizin component'e henüz hiçbir etkisi yok.

Şimdi bu etkiyi sağlayalım:

```
<input [value]="value" (keyup.enter)="val = $event.target.value; onKeyUp()">
```

Dediğimiz anda artık keyup.enter eventi trigger edilince iki iş yapılıyor, ilgili event'in elemanının yani input elemanının value'su alınır ve component içindeki val değerine aktarılır. Daha sonra da component'in onKeyUp() event'i çalışır.

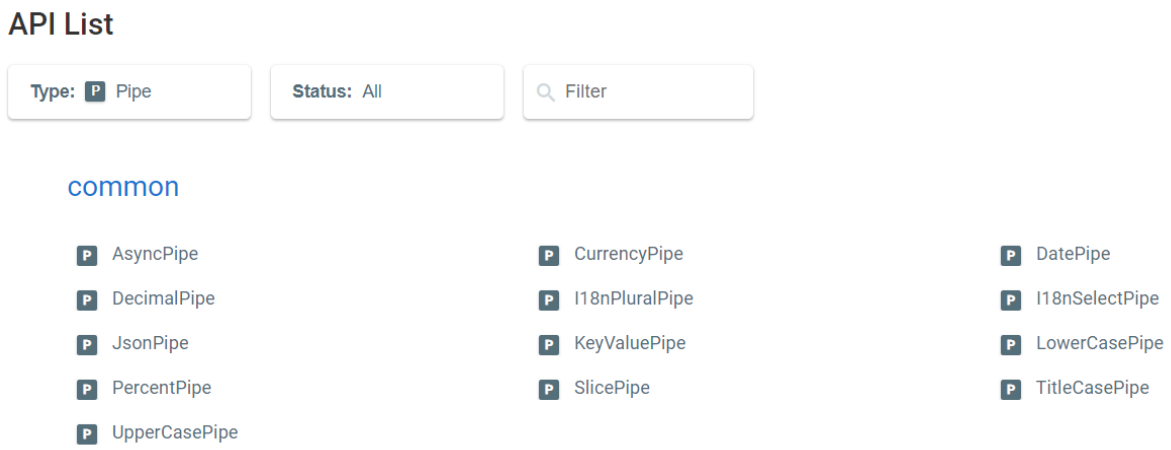
Böylece entera basılınca input bilgisi componente taşınmış oldu.

Ancak bu yol hem daha zahmetli hem de çift yönlü taşıma her zaman olmuyor, bir eventin beklenmesi lazım. ngModel daha iyi.

## 10. PIPES

Sayfalarda gösterilen verileri bazen formatlı bir şekilde göstermek isteyebiliriz. Örneğin bir tarih bilgisini gün/ay/yıl şeklinde göstermek isteyebiliriz, veya sadece dakika bilgisini çekmek isteyebiliriz. Ya da ondalıklı ifadeleri istediğimiz formatta göstermek isteyebiliriz, veya cümlelerin ilk harflerinin büyük olmasını isteyebiliriz. Tüm cümleler büyük/küçük harfle yazılsın. Fiyat bilgisi istediğimiz gibi görünsün isteyebiliriz.

İşte buna benzer format işlemleri Pipes aracılığı ile rahatlıkla yapılabilir. Angular'ın built-in pipe'larına göz atmak için angular.io>get started>api>pipe




Burada farklı pipe'ları görebiliriz. Bunlardan bazılarını örnek ile anlayalım. Bir sonraki başlık altında da custom pipe'ın nasıl yapılacağını görüp bu word'ü kapatacağız.

Örneğin component içerisine aşağıdaki gibi kullanılabilecek bazı değişkenleri tanımlayalım:

```
export class ProductComponent  {

  today: number = Date.now();
  title: string =  "Angular Kursu";
  students: number = 21536;
  price: number = 395.99;
  completed: number = 0.26;

}
```

Şimdi view içerisinde title değişkeni için 3 farklı pipe kullanalım.

```
<h4>Lower-Upper-Title Pipe</h4>
<p>{{title}}</p>
<p>{{title | lowercase}}</p>
<p>{{title | uppercase}}</p>
<p>{{title | titlecase}}</p>
```

- İlk durumda pipe kullanılmamış title olduğu gibi yazılır.
- İkincisinde lowercase olur, üçüncüsünde uppercase.
- Sonuncusunda ise her kelimenin ilk harfi büyük yazılır:

### Lower-Upper-Title Pipe

Angular Kursu

angular kursu

ANGULAR KURSU

Angular Kursu

Şimdi DatePipe'ı inceleyelim:

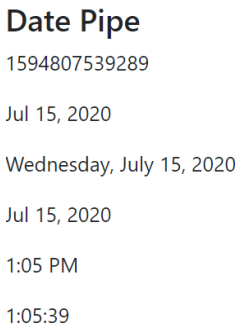
Detaylı inceleme için https://angular.io/api/common/DatePipe sayfasına bakabilirsin, burada farklı kullanımlar ile date bilgisini nasıl formatlayabileceğimiz gösterilmiş.

- `'short'`: equivalent to `'M/d/yy, h:mm a'` (`6/15/15, 9:03 AM`).
- `'medium'`: equivalent to `'MMM d, y, h:mm:ss a'` (`Jun 15, 2015, 9:03:01 AM`).
- `'long'`: equivalent to `'MMMM d, y, h:mm:ss a z'` (`June 15, 2015 at 9:03:01 AM GMT+1`).
- `'full'`: equivalent to `'EEEE, MMMM d, y, h:mm:ss a zzzz'` (`Monday, June 15, 2015 at 9:03:01 AM GMT+01:00`).
- `'shortDate'`: equivalent to `'M/d/yy'` (`6/15/15`).
- `'mediumDate'`: equivalent to `'MMM d, y'` (`Jun 15, 2015`).
- `'longDate'`: equivalent to `'MMMM d, y'` (`June 15, 2015`).
- `'fullDate'`: equivalent to `'EEEE, MMMM d, y'` (`Monday, June 15, 2015`).
- `'shortTime'`: equivalent to `'h:mm a'` (`9:03 AM`).
- `'mediumTime'`: equivalent to `'h:mm:ss a'` (`9:03:01 AM`).
- `'longTime'`: equivalent to `'h:mm:ss a z'` (`9:03:01 AM GMT+1`).
- `'fullTime'`: equivalent to `'h:mm:ss a zzzz'` (`9:03:01 AM GMT+01:00`).

Yukarıda bazı kısayollar verilmiş karşılığında da equivalent'ı yazıyor yani 'MMMM d, y' şeklinde date'i formatlamak yerine 'longDate' yazmamız yeterli oluyormuş, istersek custom bir format da oluşturabiliriz.

Şimdi component içerisinde tanımlanan date tipindeki today değişkenini datapipe ile formatlayalım:

```
<h4>Date Pipe</h4>
<p>{{today}}</p>
<p>{{today | date}}</p>
<p>{{today | date: 'fullDate'}}</p>
<p>{{today | date: 'mediumDate'}}</p>
<p>{{today | date: 'shortTime'}}</p>
<p>{{today | date: 'h:mm:ss'}}</p>
```




Bu şekilde diğer pipe işlemlerine de bakabilirsin, bence mantık anlaşıldı, gerekirse nereden bakacağını da biliyorsun sıkıntı yok.

## 11. CUSTOM PIPES

Önceki başlıkta angular'ın built-in pipe'larını anladık. Bu kısımda buna benzer pipe'ları kendimizin de yazabileceğini göreceğiz.

Yani nasıl custom pipe'lar yatayabiliriz ona bakacağız.

Diyelim ki component içerisinde bir text değişkeni içerisinde bir text'imiz olsun belki bu bir ürün description'ı olabilir ve DOM içerisinde bu text'in sadece bir kısmını göstermek isteyebiliriz, veya bu text'i istediğimiz custom bir formatta göstermek isteyebiliriz.

```
export class ProductComponent  {

 text ="Örneğin burada baya uzun bir textimiz olsun."

}
```

Şimdi app altına summary.pipe.ts isminde bir file oluşturalım ve bu bizim summary ismindeki custom pipe'ımız olacak.

```
import {Pipe, PipeTransform} from '@angular/core'

 @Pipe ({
     name: 'summary'
 })
export class SummaryPipe implements PipeTransform{

    transform(value: string) {
        if (!value) return null;
        return value.substr(0,20) + '...';
    }

}
```

Burada bir SummaryPipe class'ı yarattık. Bunu kullanabilmek için app.module içerisinde declare etmemiz gerek:

```
import { NgModule } from '@angular/core';
import { ProductComponent } from './product/product.component';
import { FormsModule } from '@angular/forms';
import { SummaryPipe } from './summary.pipe';

@NgModule({
  declarations: [
    ProductComponent,
    SummaryPipe
  ], …
```

Şimdi ilgili pipe'ı kullanmak istiyoruz diyelim, view içerisinde:

```
<p>{{text | summary}}</p>
```

Şeklinde bu pipe kullanılabiliyor, summary isminin decorator içerisinde SummaryPipe class'ına atandığına dikkat et. Ayrıca SummaryPipe içerisindeki transform içine string aldı çünkü biz text değişkenini gönderdik.

Bunun yanında başka parametreler de gönderilebilirdi.

Örneğin text'i alıp, kullanıcının istediği kadar karakterini gösteren bir pipe yazarsak:

```
import {Pipe, PipeTransform} from '@angular/core'

 @Pipe ({
     name: 'summary'
 })
export class SummaryPipe implements PipeTransform{

    transform(value: string, limit?: number) {
        if (!value) return null;
        let _limit = (limit) ? limit:20;
        return value.substr(0,_limit) + '...';
    }

}
```

Opsiyonel bir limit parametresi pipe'a gönderilsin, eğer gönderilmişse _limit değeri gönderilen limit'e eşitlensin göderilmediyse _limit = 20 olsun.

Ekstra parametreyi verebilmek için:

```
<p>{{text | summary:15}}</p>
```

Şeklinde pipe'ı kullanırız.