Tuesday: Output Property Binding

Output Property Binding

Our application at this point shows us our goals and their descriptions at the same time. Why don't we build some interactivity in it, in a way that, we don't always see the description of a goal, but instead, we click a button that toggles between showing and hiding a description? This will help us understand how output property binding works. Just as input property binding passes data into a component, **output property binding** passes data out of a component.

Let's start by adding this property to our goal blueprint to enable us to toggle between showing and hiding a goal description.

src/app/goal.ts

```
export class Goal {
  showDescription: boolean;
  constructor(public id: number,public name: string,public description: string){
    this.showDescription=false;
  }
}
```

We have changed how we create the `Goal` blueprint class by using a constructor function. We have created a property `showDescription` and assigned it to the data type `boolean`. Inside the constructor function, we have passed the goal properties we had before, as arguments of the constructor and declared that the `showDescription` property should be initialized as `false` so that the description is not displayed. We'll write the code to control this logic for hiding and showing goals later on. Notice that we have used the keyword `this` to give the `showDescription` property class access. The `public` keyword is an access modifier and it determines where the class properties are visible which in our case is anywhere outside the class. If we used the `private` keyword, the properties would only be visible inside the class.

Why use a constructor function? A constructor function enables us to define the initialization logic for creating an object. Our Goal object in this case still needs the properties `id,name` and `description` to instantiate our Goal object. The `showDescription` property, on the contrary, is not mandatory when creating a Goal object. That's the reason we have used the constructor function, we are telling our angular application that it should initialize a goal object requiring the `id,name` and `description` as mandatory properties and as well add `showDescription` to a goal object immediately setting its value to `false`. Each goal object we create from now on will, therefore, have the `showDescription` property although we will not explicitly define this property for each Goal object that we create.

Let's also change our Goals so we can put the constructor into use.

src/app/goal/goal.component.ts

```
...
export class GoalComponent implements OnInit {

  goals: Goal[] = [
    new Goal(1, 'Watch finding Nemo', 'Find an online version and watch merlin find his son'),
    new Goal(2,'Buy Cookies','I have to buy cookies for the parrot'),
    new Goal(3,'Get new Phone Case','Diana has her birthday coming up soon'),
    new Goal(4,'Get Dog Food','Pupper likes expensive snacks'),
    new Goal(5,'Solve math homework','Damn Math'),
    new Goal(6,'Plot my world domination plan','Cause I am an evil overlord'),
  ];

...
```

We have created the property `goals` and specified that it will be an array of type `Goal`. When defining each goal, we use the keyword `new` and call the Goal blueprint class, inside it specifying the three mandatory properties in the constructor function `id, name, & description`

Let's now add the logic for showing and hiding a goal description in our HTML template file.

src/app/goal/goal.component.html

```
<h1>My Goals</h1>
<hr>
<ul>
  <div *ngFor='let goal of goals;let i = index'>
    <li>{{goal.name}}</li>
    <button (click)='toggleDetails(i)'>Toggle Details</button>
    <app-goal-detail *ngIf='goal.showDescription' [goal]='goal'></app-goal-detail>
  </div>
</ul>
<p *ngIf='goals.length > 5'>You have too many goals</p>
```

In the div we have created, we have added a loop logic,`let i = index` to register the index of each goal item in the goals list. We have then displayed the goal name and below it created a button that also contains some logic.

click Event binding

The logic we have added to the button is an **event binding** syntax, the **click** event binding. We are telling the angular app to listen for a click event on this button and once it happens, it should execute the `toggleDetails()` function which we will create in a few. The `toggleDetails()` function takes in the index position of the goal item as an argument. We have then updated the *GoalDetail* template tags with the `*ngIf` directive, instructing it to display the goal description if it exists.

Show and hide Logic

Let's now create the `toggleDetails()` function that will display and hide a goal description.

src/app/goal/goal.component.ts

```
export class GoalComponent implements OnInit {
...

  toggleDetails(index){
    this.goals[index].showDescription = !this.goals[index].showDescription;
  }

  constructor() { }

  ngOnInit() {
  }

}
```

We have defined the `toggleDetails()` function in our component class and specified that it takes `index` as an argument. Inside the function, we have defined the logic for displaying the goal description, which in our case changes the `showDescription` from `false` to `true` and vice versa each time the function is executed. The `goals[index]` ensures that the function is executed for the goal at the specific index.

If our server is still running, we can now interact with our application on the web browser. We notice that the goal descriptions are no longer displayed and we have the button which toggles between displaying and hiding a goal description.