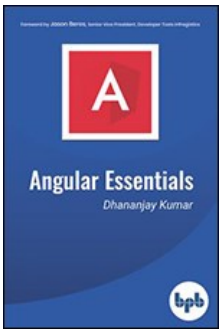

Angular Essentials: The Essential Guide to Learn Angular

by Dhananjay Kumar

BPB Publications. (c) 2019. Copying Prohibited.

Reprinted for Upendra Kumar, ACM

upendrakumar1@acm.org

Reprinted with permission as a subscription benefit of **Skillport**,

All rights reserved. Reproduction and/or distribution in whole or in part in electronic,paper or other forms without written permission is prohibited.

Chapter 2: Component and Data Binding

In this chapter, you will learn about Angular Components and various Data Binding techniques. Following topics will be covered in this chapter:

- Component
- What is Data Binding
- Interpolation
- Property Binding
- Event Binding
- Two way data binding with [(ngModel)]
- Two way data binding without [(ngModel)]

Component

In Angular applications, what you see on the browser (or elsewhere) is a component. A component consists of the following parts:

1. A TypeScript class is called Component class
2. A HTML file is called Template of the component
3. An optional CSS file for the styling of the component

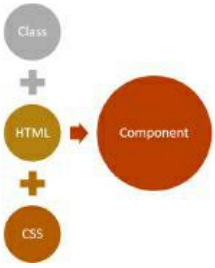

Figure 2.1

Components are type of directives, which has its own template. Whatever you see in an Angular application is a component.

Creating a Component

You can use Angular CLI command to generate a component as:

ng Generate Component Product

This command will generate **ProductComponent** as shown in *Code Listing 2.1*:

Code Listing 2.1

```
import  {  Component,  Onlnit  }  from '©angular/core';
@Component({
   selector:   'app-product',
   templateUrl:   V/product.component.html',
   styleUrls:   [`./product.component.scss']
})
export class ProductComponent implements Onlnit  {

constructor()   {  }

ngOnInit()   { }
```


Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.

```
}
```

A component is a class decorated with @**Component** decorator. There are mainly four steps to create a component:

1. Create a class and export it. This class will contain data and the logic.
2. Decorate the class with @**component** metadata. Metadata describes the component and sets the value for different properties.
3. Import the required libraries and modules to create the component.
4. Create template of the component and optionally style of the component.

As you can see, generated **ProductComponent** consists of:

- A class to hold data and the logic;
- HTML template and styles to display data in the app. It is also called as a view, which is seen by the user on the screen to interact.
- Metadata, which defines the behavior of a component. Component metadata is applied to the class using the @Component decorator. Different behaviors of the component can be passed as properties of the object, which is an input parameter of the @Component decorator.

Component Metadata

@Component decorator decorates a class as a component. It is a function, which takes an object as a parameter. In the @Component decorator, we can set the values of different properties to set the behavior of the component. The most used properties are as follows:

- template
- templateUrl
- Providers
- styles
- styleUrls
- selector
- encapsulation
- changeDetection
- animations
- viewProviders

Apart from the above mentioned properties, there are other properties also. Now let us look into these important properties one by one.

Template and TemplateUrl

A template is the part of the component which gets rendered on the page. We can create a template in two possible ways:

1. Inline template : template property
2. Template in an external file : templateUrl property

To create inline template **tilt** symbol is used to create multiple lines in the template. A single line inline template can be created using either single quotes or double quotes. For inline template set value of template property. Complex template can be created in an external HTML file and can be set using templateUrl property.

Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.

Selector

A component can be used using the selector. In the above example, the selector property is set to **<app-product>.** We can use the component on template of other components using its selector.

Styles and StyleUrls

A component can have its own styles or it can refer to various other external style sheets. To work with styles, @Component metadata has styles and styleUrls properties. We can create inline styles by setting the value of the styles property. We can set external style using styleUrls property.

Providers

To inject a service in a component, you pass that to providers array. Component metadata has an array type property called the provider. In the providers, we pass a list of services being injected in the component. We will cover this in detail in further sections.

ChangeDetection

This property determines how change detector will work for the component. We set **`ChangeDetectionStrategy`** of the component in the property. There are two possible values:

1. Default
2. onPush

We will cover this property in detail in further sections.

Encapsulation

This property determines whether Angular will create shadow DOM for component or not. It determines **`ViewEncapsulation`** mode of the component. There are four possible values:

1. Emulated this is default
2. Native
3. None
4. ShadowDom

Template

When you generate a component using Angular CLI, by default selector, **`templateUrl`**, and **`styleUrl`** properties are set. For **`ProductComponent`** template is in external HTML file **`product.component.html`** as shown in *Code Listing 2.2.*

Code Listing 2.2

```
<P>
   product works!
</p>
```

You can pass data and capture events between component class and its template using Data Binding. We will cover this in detail, in further sections.

Using a Component

Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.

A component can be used inside an Angular application in various ways:

- As a root component.
- As a child component. We can use a component inside another component.
- Navigate to a component using Routing. In this case, component will be loaded in **RouterOutlet.**
- Dynamically loading component using **ComponentFactoryResolver.**

  Component must be part of a module and to use a component in a module, first import that and then pass it to declaration array of the module. Refer *Code Listing 2.3*

Code Listing 2.3

```
@NgModule({
   declarations:   [
      AppComponent,
      ProductComponent
],
```

Data Binding

In Angular, Data Binding determines how data will flow in between Component class and Component Template.

Angular provides us with the three types of data bindings. They are as follows:

- Interpolation
- Property Binding
- Event Binding

Figure 2.2

Let's see each, one by one.

Interpolation

Angular interpolation is one-way data binding. It is used to pass data from component class to the template. The syntax of interpolation is `{{propertyname}}` .

Let's say, we have component class as shown in code listing 2.4:

Code Listing 2.4

```
export class AppComponent {

      product = {
         title:   'Cricket Bat',
         price:  500
      };
   }
```


Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.

We need to pass the product from the component class to the template. Keep in mind that to keep example simple, I'm hard coding the value of the product object, however, in area! scenario, data could be fetched from the database using the API. We can display value of the product object using interpolation, as shown in the **code listing 2.5:**

Code Listing 2.5

```
  <h1>Product</h1>
  <h2>Title : {{product.title}}</h2>
  <h2>Price  :  {{product.price}}</h2>
```

Using interpolation, data is passed from the component class to the template. Ideally, whenever the value of the product object is changed, the template will be updated with the updated value of the product object.

In Angular, there is something called **`ChangeDetector`** Service, which makes sure that value of property in the component class and the template are in sync with each other.

Therefore, if you want to display data in Angular, you should use interpolation data binding.

Property Binding

Angular provides you with a second type of binding called *Property Binding.* The syntax of property binding is the square bracket **`[]`**. It allows to set the property of HTML elements on a template with the property from the component class.

So, let's say that you have a component class like *Code Listing 2.6.*

Code Listing 2.6

```
      export class AppComponent {
      btnHeight = 100; btnWidth = 100;
   }
```

Now, you can set height and width properties of a button on template with the properties of the component class using the property binding as shown in *Code Listing 2.7.*

Code Listing 2.7

```
<button
   [style.height.px]  =  'btnHeight'
      [style.width.px]   =  'btnWidth'>
         Add Product
</button >
```

Angular Property Binding is used to set the property of HTML Elements with the properties of the component class. You can also set properties of other HTML elements like image, list, table, etc. Whenever the property's value in the component class changes, the HTML element property will be updated in the property binding.

Event Binding

Angular provides you the third type of binding to capture events raised on template in a component class. For instance, there's a button on the component template and, on the click of the button, you want to call a function in component class. You can do this using Event Binding. The syntax behind Event Binding is **(eventname).**

For this example, you might have a component class like this as shown in *Code Listing 2.8.*

Code Listing 2.8

```
export class AppComponent {
      addProduct()   {
         console.log(`add product');
      }
   }
```


Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.

You want to call **`addProduct`** function on the click of the button on the template. You can do this using event binding: like *Code Listing 2.9.*

Code Listing 2.9

```
<h1>Product</h1>
<button  (click)='addProduct()'>
      Add Product
</button>
```

Angular provides you these three bindings. In event binding, data flows from template to class and, in property binding and interpolation, data flows from class to template. Refer *figure 2.3:*

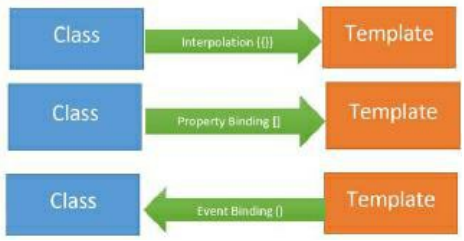

Figure 2.3

Two-Way Data Binding

Angular does not have built-in two-way data binding, however, by combining Property Binding and Event Binding, you can achieve Two-Way Data Binding.

Figure 2.4

Angular provides us a directive, ngModel, to achieve two-way data binding, and It's very easy to use. First, import **Forms** Module and then you can create two-way data binding as shown in *Code Listing 2.10.*

Code Listing 2.10

```
export class AppComponent   {

      name  =   'foo';
 }
```

We can, in two-way, data bind the name property with an input box as shown in *Code Listing 2.11.*

Code Listing 2.11

```
<input type="text"   [(ngModel)]='name'  />

<h2>{{name}}</h2>
```

As you see, we are using **`[(ngModel)]`** to create two-way data binding in between input control and name property. Whenever a user changes the value of the input box, the name property will be updated and vice versa.

Two-Way Data Binding without ngModel

To understand **`ngModel`** directive working, let us see how we can achieve two-way data binding without using **`ngModel`** directive. To do that, we need to use the following:

- Property binding to bind expression to value property of the input element. In this demo, we are binding name variable expression to value property.

Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.

- Event binding to emit input event on the input element. Yes, there is an input event which will be fired whenever user will input to the input element. Using event binding, input event would be bind to an expression.

So, using the property binding and the event binding, two-way data binding can be achieved as shown in the *Code Listing 2.12.*

Code Listing 2.12

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

@Component({
   selector:   'app-root',
   template:  '
      <div class="container">
       <input   [value]="name"   (input)="name=$event.
target.value" />
         <br/>
         <h1>Hello {{name}}</h1>
      </div>
      '
})
export class AppComponent {
      name =  '';
}
```

Just like **`ngModel`** directive demo, in this demo also, when typing into the input element, the input element's value will be assigned to name variable and also it would be displayed back to the view.

Let us understand few important things here:

- **`[value]="name"`** is the property binding. We are binding value property of the input element with variable (or expression) name.
- **`(input)= "expression"`** is event binding. Whenever input event will be fired expression will be executed.
- **`'name=$event.target.value'`** is an expression which assigns entered value to name variable.
- Name variable can be accessed inside **`AppComponent`** class.

Summary

In this chapter, we learnt about Angular Components and various Data Binding techniques. A good understanding of data bindings is important to leverage the other features of Angular. In this chapter you learnt about following topics:

- Component
- What is Data Binding
- Interpolation
- Property Binding
- Event Binding
- Two way data binding with **`[(ngModel)]`**
- Two way data binding without **`[(ngModel)]`**

Reprinted for ACM/upendrakumar1, ACM
BPB Publications (c) 2019, Copying Prohibited.