Angular 8 : Online Class

Class – 7

By: Sahosoft Solutions

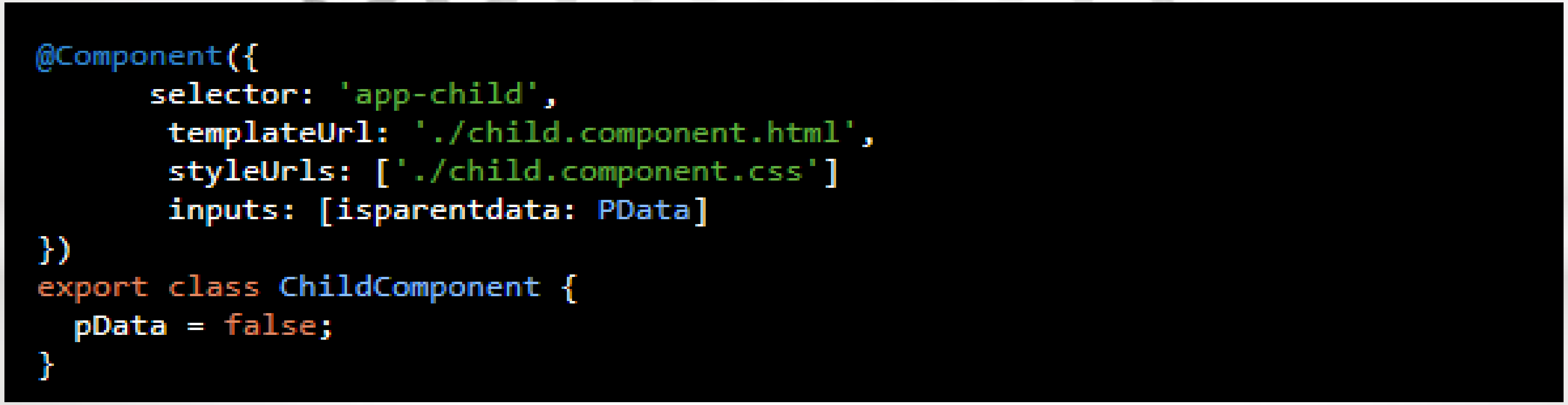
Presented by : Chandan Kumar

Input Property

Input property is used within one component (child component) to receive a value from another component (parent component). This is a one-way communication from parent to child. A component can receive a value from another component through component input property.

Aliasing Input Property

Input property can alias the name of the input property. Means You can alias our input property so that it is not linked to the actual variable name, we can do it as shown below:



```
@Component({
      selector: 'app-child',
       templateUrl: './child.component.html',
       styleUrls: ['./child.component.css']
       inputs: [isparentdata: PData]
})
export class ChildComponent {
  pData = false;
}
```

Output Property

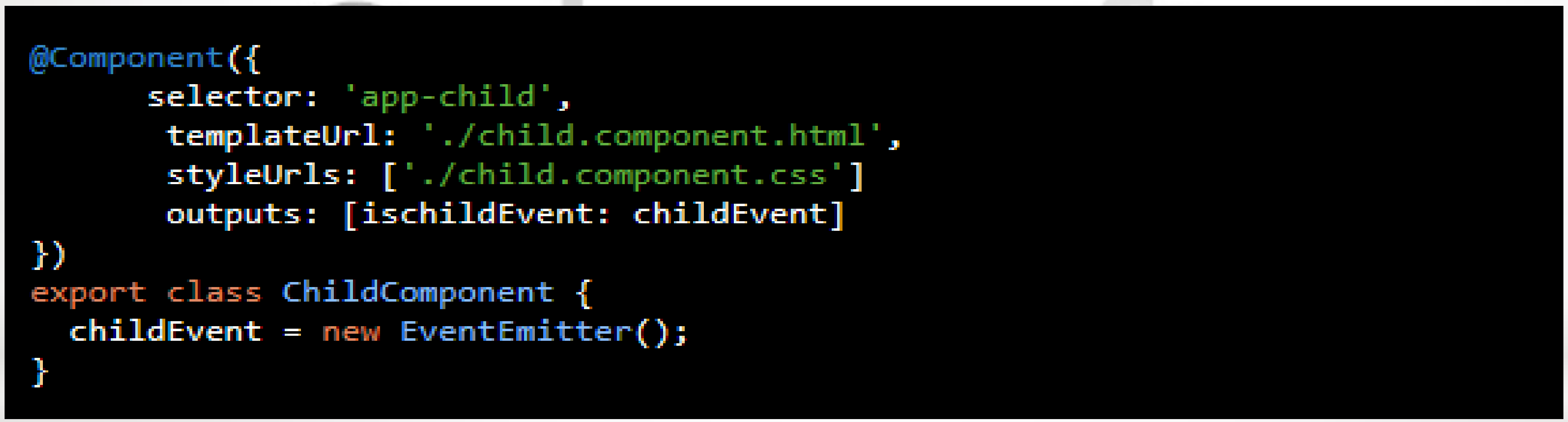
Input property is used send data from one component (child component) to calling component (parent component). This is a one-way communication from child to parent. This property name becomes a custom event name for the calling component. The output property can also create an alias with the property name as Output (alias) and now this alias name will be used in the custom event link in the call component. Now we will see how to use the output property.

It is the topic of the **Component Interaction** in angular. As we know, the angular application is based on small components, so it is very difficult to pass data from a child component to a parent component, in this scenario the output property is very useful. The angular components have a better way of notifying the parent components that something has changed through events. The "outputs" identify events that a component can fire to send information from the hierarchy to its parent from its child component.

Output Property

Aliasing Output Property:

Output property can alias the name of the output property. Means You can alias our output property so that it is not linked to the actual variable name, we can do it as shown below:



```
@Component({
      selector: 'app-child',
       templateUrl: './child.component.html',
       styleUrls: ['./child.component.css']
       outputs: [ischildEvent: childEvent]
})
export class ChildComponent {
  childEvent = new EventEmitter();
}
```

Providers Property

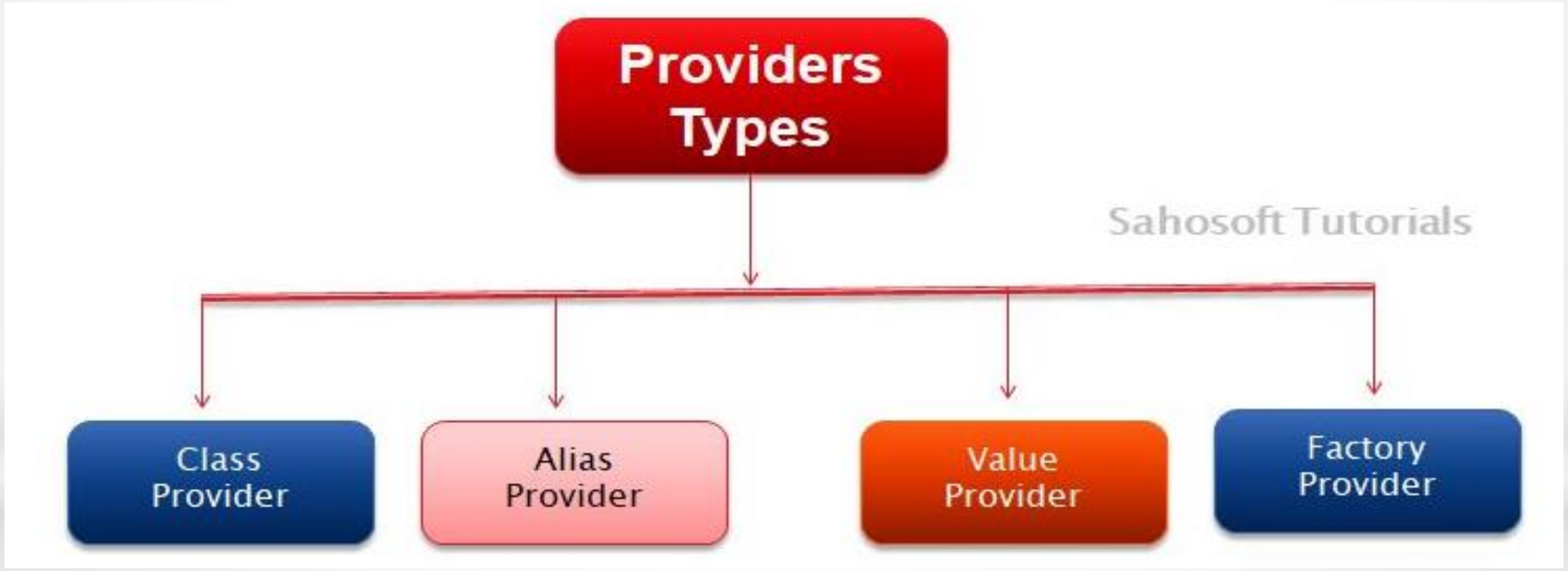
providers metadata is available in the @NgModule() decorator and @Component() decorator. provider's metadata is used to configure the provider.

A provider provides concrete and runtime version of a dependency value. The injector injects the objects provided by provider into components and services. Therefore, it is necessary to configure a service with the provider, otherwise the injector will not be able to inject it.

The injector injects the objects provided by provider into components and services. Only those classes configured by providers are available for dependency injection (DI).

Providers Property

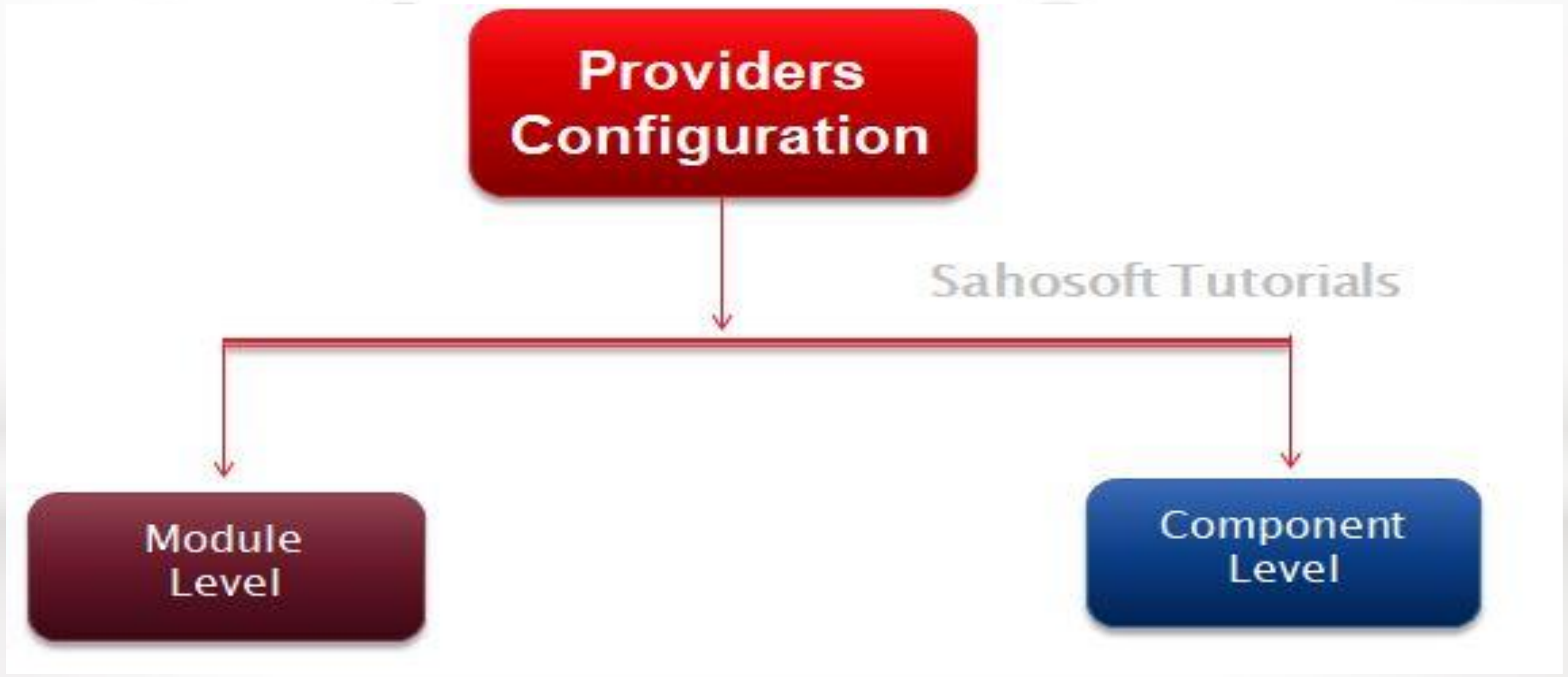
Angular offers different types of providers as a class provider, alias provider, value provider and factory provider as shown below.



Providers Property

Injector creates a singleton object of a class configured by providers.
Providers can be configured at Two level as shown below.

- Module level
- Component level



Providers Property

Module level:

Providers can be configured at the module level. when providers of @NgModule() decorator configure a service in the application module, that service will be available for injection of dependencies on all those components that are configured in the metadata declarations of that decorator @NgModule().
In short, if a service is configured using the provider in the module, it will be available for all components configured in module for dependency injection.

Providers Property

Component level:

Providers can be configured at the component level. when @Component() decorator providers configure a service in a component, that service will be available for injecting dependencies into the current component and its secondary components up to the bottom component.

In short, if a service is configured using the provider in the component, it will be available in the current component and its children component up to the bottom component.

Suppose that in our application, if we have configured a service using the providers in the root component (app.component.ts), it will be available in all the components and services of the application for injecting dependencies.

Providers Property

Component level:

Providers can be configured at the component level. when @Component() decorator providers configure a service in a component, that service will be available for injecting dependencies into the current component and its secondary components up to the bottom component.

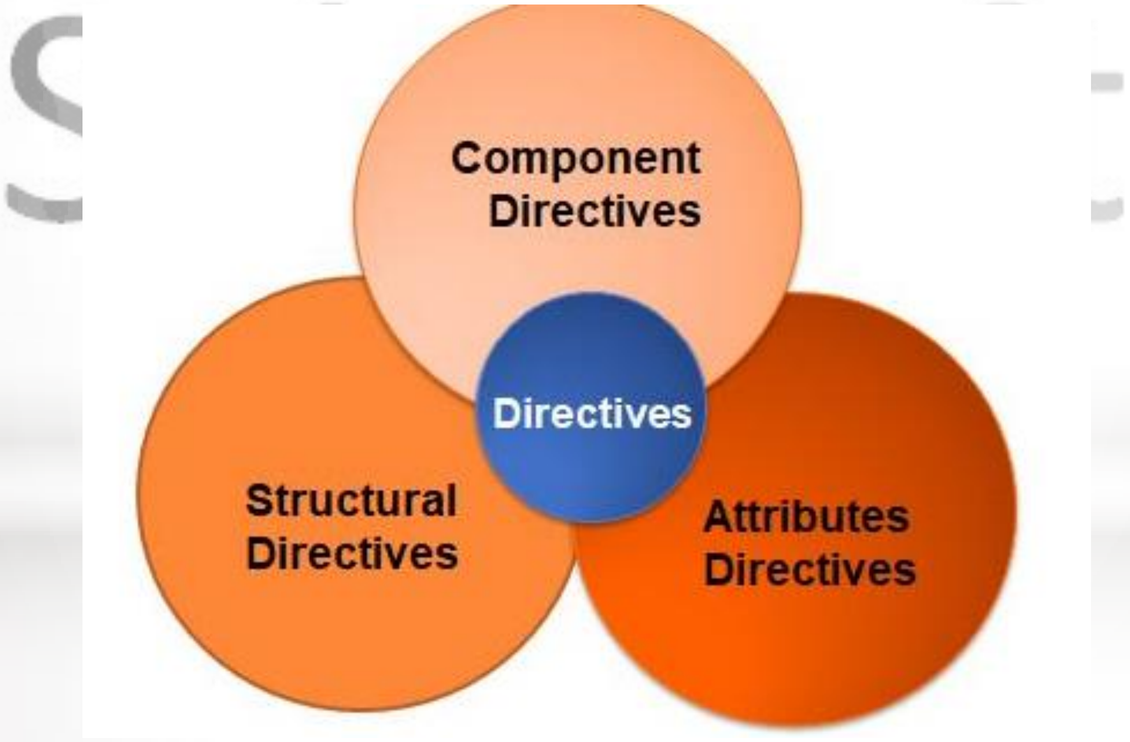
In short, if a service is configured using the provider in the component, it will be available in the current component and its children component up to the bottom component.

Suppose that in our application, if we have configured a service using the providers in the root component (app.component.ts), it will be available in all the components and services of the application for injecting dependencies.

Directive

Basically, directives are used to extend the power of the HTML attributes and to change the appearance or behavior of a DOM element.

Directive in Angular is a javascript class, which is declared as @directive. Angular has 3 types of directives, which are listed below –



Structural directives

Structural directives are a key part of Angular everyone should be familiar with. They are responsible for manipulating DOM through adding, removing or changing the elements. Even if you have never written a structural directive yourself, you have probably been using *ngIf and *ngFor in your templates pretty often. The asterisk (*) states it is a structural directive.

In other words, Structural Directives are directives which change the structure of the DOM by adding or removing its elements. **Structural directives** reconstruct the layout by adding, removing, and replacing elements in **DOM**. The **structural directives** are responsible for shape or reshape the DOM's structure, typically by adding, deleting, or modifying elements. Similar to the other directives, you apply the structural directive to a host element. The directive then performs whatever it is intended to do with that host element. The structural directives are very simple to recognize. An **asterisk (*)** precedes the directive attribute name. It does not require brackets or parentheses like attribute directive. These directives are the way to handle how the component or the element renders in a template.

Structural directives

There are basically 3 built in structural directives available in Angular.

- NgIf (*ngIf)
- NgFor (*ngFor)
- NgSwitch (*ngSwitch)

NgIf

Here we will understand using NgIf with HTML elements. NgIf is an angular directive that is used to add an element subtree for **true** value of expression. NgIf is used as *ngIf="expression". Here if "expression" value is either **false** or **null**, in both cases the element subtree will not be added to DOM.