20 important Angular Interview questions with answers

by

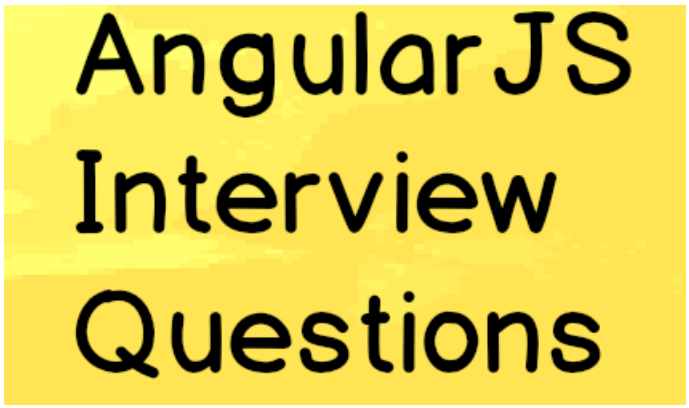
http://www.questpond.com

Contents

Angular Interview questions and answers

AngularJS is one of those hot topics which interviewer's ask for Web programming. In this article we will run through some important Interview questions around AngularJS and how we should be go about answering the same.

Do not forget to see our Learn MVC with Angular in 2 days i.e. (16 hours) video series. Start from this youtube video link. https://www.youtube.com/watch?v=Lp7nSImO5vk



What is AngularJS ?

"AngularJS is a JavaScript framework which simplifies binding JavaScript objects with HTML UI elements."

Let us try to understand the above definition with simple sample code.

Below is a simple "Customer" function with "CustomerName" property. We have also created an object called as "Cust" which is of "Customer" class type.

```
function Customer()
{
      this.CustomerName = "AngularInterview";
}
var Cust = new Customer();
```

Now let us say the above customer object we want to bind to a HTML text box called as "TxtCustomerName". In other words when we change something in the HTML text box the customer object should get updated and when something is changed internally in the customer object the UI should get updated.

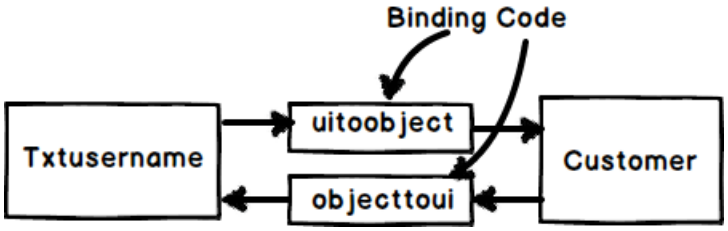
```
<input type=text id="TxtCustomerName" onchange="UitoObject()"/>
```

So in order to achieve this communication between UI to object developers end up writing functions as shown below. "UitoObject" function takes data from UI and sets it to the object while the other function "ObjecttoUI" takes data from the object and sets it to UI.

```
function UitoObject()
{
      Cust.CustomerName = $("#TxtCustomerName").val();
}
```

```
function ObjecttoUi()
{
$("#TxtCustomerName").val(Cust.CustomerName);
}
```

So if we analyze the above code visually it looks something as shown below. Your both functions are nothing but binding code logic which transfers data from UI to object and vice versa.

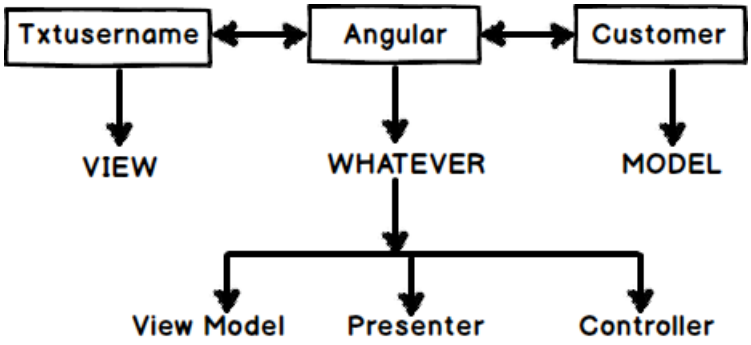


Now the same above code can be written in Angular as shown below. The javascript class is attached to a HTML parent div tag using "ng-controller" directive and the properties are binded directly to the text box using "ng-model" declarative.

So now whatever you type in the textbox updates the "Customer" object and when the "Customer" object gets updated it also updates the UI.

```
<div ng-controller="Customer">
        <input type=text id="txtCustomerName"  ng-model="CustomerName"/>
</div>
```

In short if you now analyze the above code visually you end up with something as shown in the below figure.You have the VIEW which is in HTML, your MODEL objects which are javascript functions and the binding code in Angular.



Now that binding code have different vocabularies.

- Some developers called it "ViewModel" because it connects the "Model" and the "View" .
- Some call it "Presenter" because this logic is nothing but presentation logic.
- Some term it has "Controller" because it controls how the view and the model will communicate.

To avoid this vocabulary confusion Angular team has termed this code as “Whatever”. It’s that “Whatever” code which binds the UI and the Model. That’s why you will hear lot of developers saying Angular implements “MVW” architecture.

Explain Directives in Angular?

Directives are attributes decorated on the HTML elements. All directives start with the word “ng”. As the name says directive it directs Angular what to do.

For example below is a simple “ng-model” directive which tells angular that the HTML textbox “txtCustomerName” has to be binded with the “CustomerName” property.

```
<input type=text id="txtCustomerName"  ng-model="CustomerName"/>
```

Some of the most commonly used directives are ng-app,ng-controller and ng-repeat.

What are controllers and need of ng-controller and ng-model in Angular?

“Controllers” are simple javascript function which provides data and logic to HTML UI. As the name says controller they control how data flows from the server to HTML UI.

For example below is simple “Customer” controller which provides data via “CustomerName” and “CustomerCode” property and Add/ Update logic to save the data to database.

Note: - Do not worry too much about the $scope , we will discuss the same in the next question.

```
function Customer($scope)
{
        $scope.CustomerName = "Shiv";
        $scope.CustomerCode = "1001";
        $scope.Add = function () {
        }
        $scope.Update = function () {
        }
}
```

“ng-controller” is a directive.Controllers are attached to the HTML UI by using the “ng-controller” directive tag and the properties of the controller are attached by using “ng-model” directive. For example below is a simple HTML UI which is attached to the “Customer” controller via the “ng-controller” directive and the properties are binded using “ng-model” directive.

```
<div ng-controller="Customer">
        <input type=text id="CustomerName"  ng-model="CustomerName"/><br />
        <input type=text id="CustomerCode"  ng-model="CustomerCode"/>
</div>
```

What are expressions in Angular?

Angular expressions are unit of code which resolves to value. This code is written inside curly braces "{".

Below are some examples of angular expressions:-

The below expression adds two constant values.

```
{{1+1}}
```

The below expression multiplies quantity and cost to get the total value.

```
The value total cost is {{ quantity * cost }}
```

The below expression displays a controller scoped variable.

```
<div ng-controller="CustomerVM">
      The value of Customer code is {{CustomerCode}}
</div>
```

How can we initialize Angular application data?

We can use "ng-init" directive to achieve the same. You can see in the below example we have used "ng-init" directive to initialize the "pi" value.

```
<body ng-app="myApp" ng-init="pi=3.14">
The value of pi is {{pi}}
</body>
```

Explain $scope in Angular?

"$scope" is an object instance of a controller. "$scope" object instance get's created when "ng-controller" directive is encountered.

For example in the below code snippet we have two controllers “Function1” and “Function2”. In both the controllers we have a “ControllerName” variable.

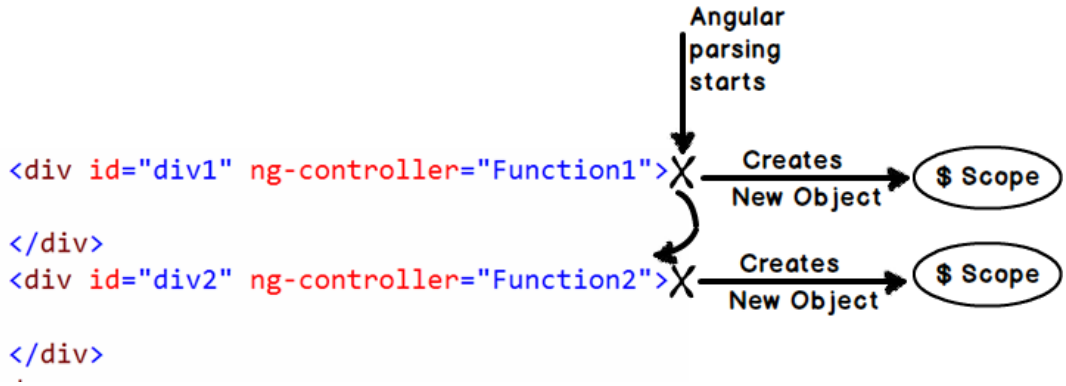
```
function Function1($scope)
{
$scope.ControllerName = "Function1";
}
function Function2($scope)
{
$scope.ControllerName = "Function2";
}
```

Now to attach the above controllers to HTML UI we need to use “ng-controller” directive. For instance you can see in the below code snippet how “ng-controller” directive attaches “function1” with “div1” tag and “function2” with “div2” tag.

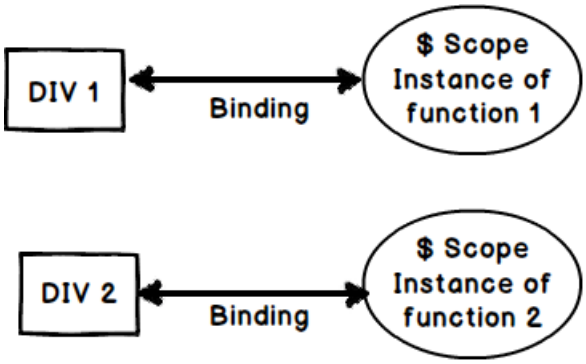
```
<div id=”div1” ng-controller="Function1">
      Instance of {{ControllerName}} created
</div>
<div id=”div2” ng-controller="Function2">
      Instance of {{ControllerName}} created
</div>
```

So this is what happens internally. Once the HTML DOM is created Angular parser starts running on the DOM and following are the sequence of events:-

- The parser first finds “ng-controller” directive which is pointing to “Function1”. He creates a new instance of “$scope” object and connects to the “div1” UI.
- The parser then starts moving ahead and encounters one more “ng-controller” directive which is pointing to “Function2”. He creates a new instance of “$scope” object and connects to the “div2” UI.

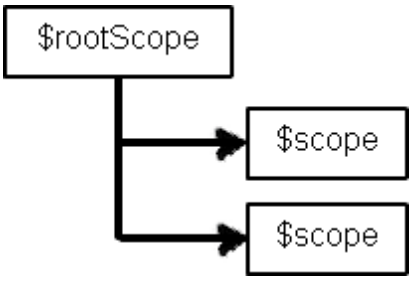


Now once the instances are created, below is a graphical representation of the same. So the "DIV1" HTML UI is binded with "function1" $scope instance and the "DIV2" HTML UI is binded with "function2" $scope instance. In other words now anything changes in the $scope object the UI will be updated and any change in the UI will update the respective $scope object.

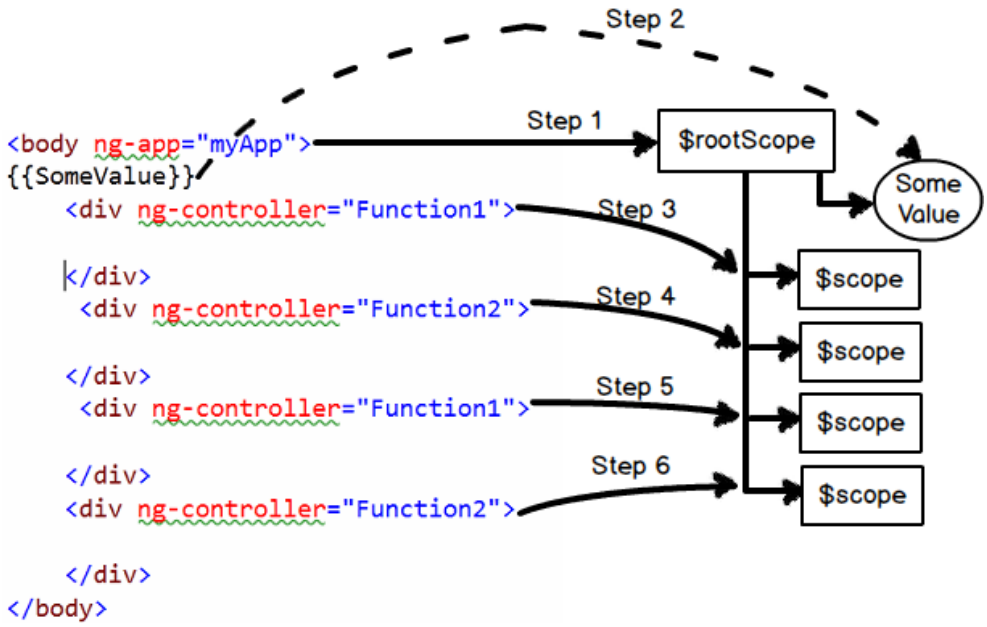


What is "$rootScope" and how is it related with "$scope"?

"$rootScope" is a parent object of all "$scope" angular objects created in a web page.



Let us understand how Angular does the same internally. Below is a simple Angular code which has multiple "DIV" tags and every tag is attached to a controller. So let us understand step by step how angular will parse this and how the "$rootScope" and "$scope" hierarchy is created.



The Browser first loads the above HTML page and creates a DOM (Document object model) and Angular runs over the DOM. Below are the steps how Angular creates the rootscope and scope objects.

- Step 1:- Angular parser first encounters the "ng-app" directive and creates a "$rootScope" object in memory.
- Step 2:- Angular parser moves ahead and finds the expression {{SomeValue}}. It creates a variable from the expression and attaches this variable to the "$rootScope" object created in Step 1.
- Step 3:- Parser then finds the first "DIV" tag with "ng-controller" directive which is pointing to "Function1" controller. Looking at the "ng-controller" directive it creates a "$scope" object instance for "Function1" controller. This object it then attaches to "$rootScope" object.
- Step 4:- Step 3 is then repeated by the parser every time it finds a "ng-controller" directive tag. Step 5 and Step 6 is the repetition of Step 3.

If you want to test the above fundamentals you can run the below sample Angular code. In the below sample code we have created controllers "Function1" and "Function2". We have two counter variables one at the root scope level and other at the local controller level.

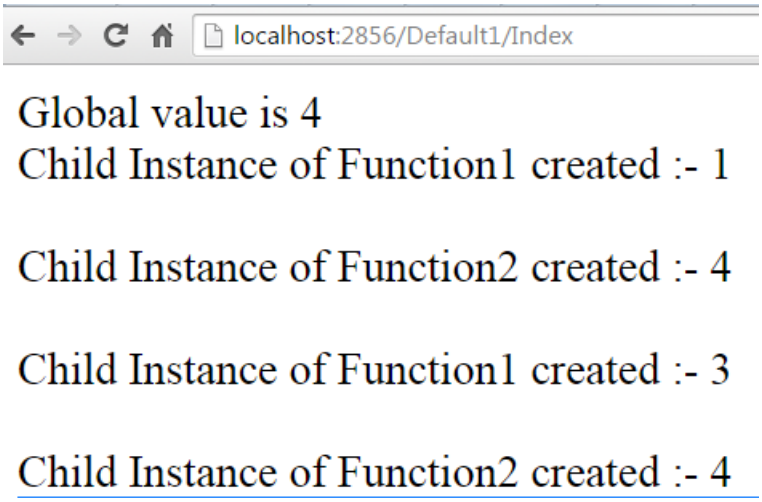
```
<script language=javascript>
function Function1($scope, $rootScope)
{
        $rootScope.Counter = (($rootScope.Counter || 0) + 1);
        $scope.Counter = $rootScope.Counter;
        $scope.ControllerName = "Function1";
}
function Function2($scope, $rootScope)
{
```

```
        $rootScope.Counter = (($rootScope.Counter || 0) + 1);
        $scope.ControllerName = "Function2";
}
    var app = angular.module("myApp", []); // creating a APP
    app.controller("Function1", Function1); // Registering the VM
    app.controller("Function2", Function2);

</script>
```

Below is the HTML code for the same. You can we have attached “Function1” and “Function2” two times with “ng-controller” which means four instances will be created.

```
<body ng-app="myApp" id=1>
   Global value is {{Counter}}<br />
    <div ng-controller="Function1">
       Child Instance of {{ControllerName}} created :- {{Counter}}
    </div>    <br />
     <div ng-controller="Function2">
       Child Instance of {{ControllerName}} created :- {{Counter}}
    </div>    <br />
     <div ng-controller="Function1">
        Child Instance of {{ControllerName}} created :- {{Counter}}
    </div> <br />
    <div ng-controller="Function2">
        Child Instance of {{ControllerName}} created :- {{Counter}}
    </div>    <br />
</body>
```



Above is the output of the code you can see the global variable of root scope has be incremented four times because four instances of $scope have been created inside "$rootScope" object.

Do I need Jquery for Angular?

No , you do not need Jquery for Angular. It's independent of Jquery.

How is the data binding in Angular ?

Its two way binding. So whenever you make changes in one entity the other entity also gets updated.

How do we make HTTP get and post calls in Angular?

To make HTTP calls we need to use the "$http" service of Angular. In order to use the http services you need to make provide the "$http" as a input in your function parameters as shown in the below code.

```
function CustomerController($scope,$http)
{
      $scope.Add = function()
      {
            $http({ method: "GET", url: "http://localhost:8438/SomeMethod"
}).success(function (data, status, headers, config)
            {
                  // Here goes code after success
            }
      }
}
```

"$http" service API needs atleast three things:-

- First what is the kind of call "POST" or "GET".
- Second the resource URL on which the action should happen.
- Third we need to define the "success" function which will be executed once we get the response from the server.

```
$http({ method: "GET", url: "http://localhost:8438/SomeMethod"
}).success(function (data, status, headers, config)
{
  // Here goes code after success
}
```

How do we pass data using HTTP POST in Angular ?

You need to pass data using the "data" keyword in the "$http" service API function. In the below code you can see we have created a javascript object "myData" with "CustomerName" property. This object is passed in the "$http" function using HTTP POST method.

```
Var myData = {};
myData.CustomerName = "Test";
$http({ method: "POST",
      data: myData,
      url: "http://www.xyz.com"})
      .success(function (data, status, headers, config)
      {
        // Here goes code after success
      }
```

What is dependency injection and how does it work in Angular?

Dependency injection is a process where we inject the dependent objects rather than consumer creating the objects. DI is everywhere in Angular or we can go one step ahead and say Angular cannot work without DI.

For example in the below code "$scope" and "$http" objects are created and injected by the angular framework. The consumer i.e. "CustomerController" does not create these objects himself rather Angular injects these objects.

```
function CustomerController($scope,$http)
{
// your consumer would be using the scope and http objects
}
```

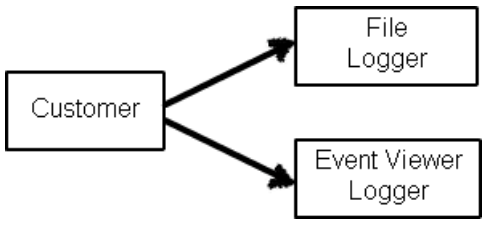
How does DI benefit in Angular?

There are two big benefits of DI: - Decoupling and Testing.

Let's first start with Decoupling. Consider your application has a logger functionality which helps to log errors , warning etc in some central place. This central place can be a file, event viewer, database etc.

```
function FileLogger()
{
        this.Log = function () {
            alert("File logger");
        };
}
function EventLogger()
{
        this.Log = function () {
            alert("Event viewer logger");
        };
}
```

Now let's say you have a "Customer" class who wants to use the "Logger" classes. Now which "Logger" class to use depends on configuration.



So the code of "Customer" is something as shown below. So depending on the configuration "Customer" class either creates "FileLogger" or it creates "EventLogger" object.

```
function Customer($scope, Logger)
{
        $scope.Logger = {};
        if (config.Loggertype = "File")
        {
            $scope.Logger = new FileLogger();
        }
        else
       {
            $scope.Logger = new EventLogger();
        }
}
```

But with DI our code becomes something as shown below. The “Customer” class says he is not worried from where the “Logger” object comes and which type of “Logger” objects are needed .He just wants to use the “Logger” object.

```
function Customer($scope,$http, Logger)
{
        $scope.Logger = Logger;
}
```

With this approach when a new “Logger” object gets added the “Customer” class does not have to worry about the new changes because the dependent objects are injected by some other system.
The second benefit of DI is testing. Let’s say you want to test the “Customer” class and you do not have internet connection. So your “$http” object method calls can throw errors. But now you can mock a fake “$http” object and run your customer class offline without errors.The fake object is injected using DI.

What are services in Angular?

Service helps to implement dependency injection. For instance let’s say we have the below “Customer” class who needs “Logger” object. Now “Logger” object can be of “FileLogger” type or “EventLogger” type.

```
function Customer($scope,$http, Logger)
{
```

```
        $scope.Logger = Logger;
}
```

So you can use the “service” method of the application and tie up the “EventLogger” object with the “Logger” input parameter of the “Customer” class.

```
var app = angular.module("myApp", []); // creating a APP
app.controller("Customer", Customer); // Registering the VM
app.service("Logger", EventLogger); // Injects a global Event logger object
```

So when the controller object is created the “EventLogger” object is injected automatically in the controller class.

Are Service object instances global or local?

Angular Services create and inject global instances. For example below is a simple “HitCounter” class which has a “Hit” function and this function increments the variable count internally every time you call hit the button.

```
function HitCounter()
{
       var i = 0;
        this.Hit = function ()
        {
            i++;
            alert(i);
        };
}
```

This “HitCounter” class object is injected in “MyClass” class as shown in the below code.

```
function MyClass($scope, HitCounter)
{
      $scope.HitCounter = HitCounter;
}
```

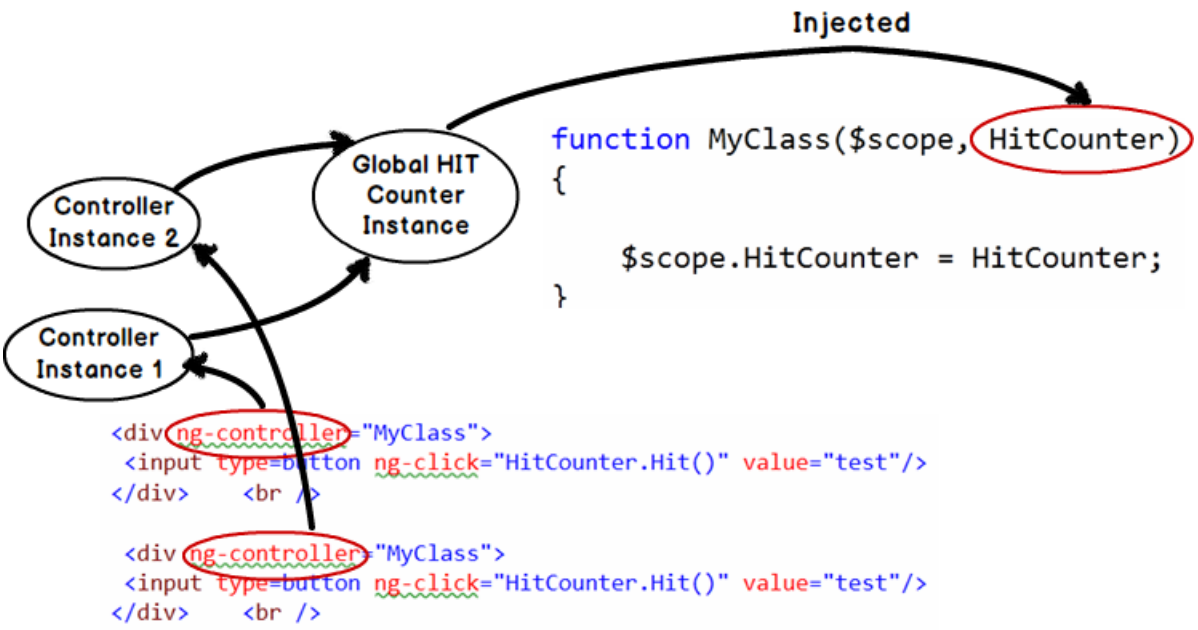
Below code advises the Angular framework to inject “HitCounter” class instance in the “MyClass” class. Read the last line of the below code specially which says to inject the inject the “HitCounter” instance.

```
var app = angular.module("myApp", []); // creating a APP
app.controller("MyClass", MyClass); // Registering the VM
app.service("HitCounter", HitCounter); // Injects the object
```

Now let's say that the "Controller" "MyClass" is attached to two div tag's as shown in the below figure.

So two instances of "MyClass" will be created. When the first instance of "MyClass" is created a "HitCounter" object instance is created and injected in to "MyClass" first instance.

When the second instance of "MyClass" is created the same "HitCounter" object instance is injected in to second instance of "MyClass".
Again I repeat the same instance is injected in to the second instance, new instances are not created.

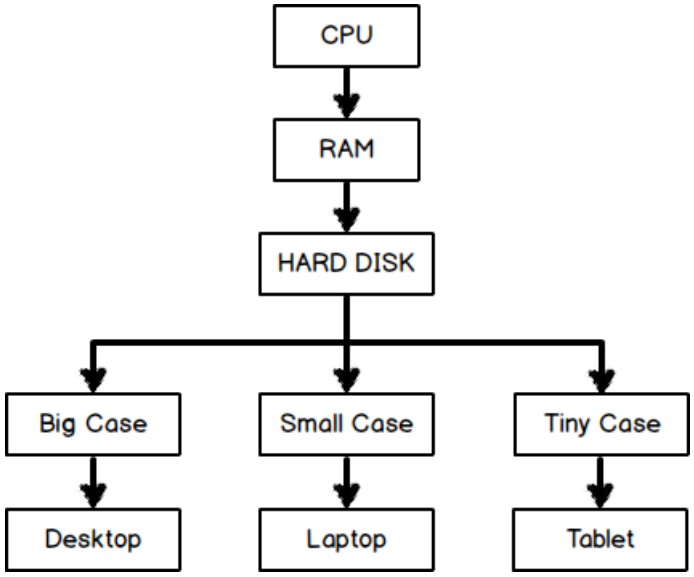


If you execute the above code you will see counter values getting incremented even if you are coming through different controller instances.

What is a Factory in Angular?

"Factory" in real world means a premise where products are manufactured. Let's take an example of a computer manufacturing firm. Now the company produces different kinds and sizes of computers like laptops, desktops, tablets etc.

Now the process of manufacturing the computer products are same with slight variation. To manufacture any computer we need processor, RAM and hard disk. But depending on what kind of final case packing is the final product shapes.



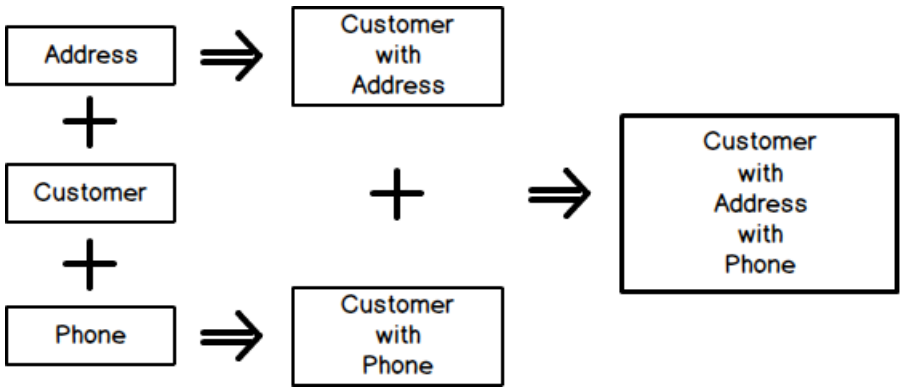
That's what the use of Factory in Angular.

For example see the below code we have a "Customer", "Phone" and "Address" class.

```
function Customer()
{
        this.CustomerCode = "1001";
        this.CustomerName = "Shiv";
}
function Phone()
{
        this.PhoneNumber = "";
}
function Address()
{
        this.Address1 = "";
        this.Address2 = "";
}
```

So now we would create different types of “Customer” object types using the combination of “Address” and “Phones” object.

- We would like to combine “Customer” with “Address” and create a “Customer” object which has “Address” collection inside it.
- Or must be we would like to create “Customer” object with “Phone” objects inside it.
- Or must be “Customer” object with both “Phone” and “Address” objects.



In other words we would like to have different permutation and combination to create different types of “Customer” objects.

So let’s start from bottom. Let’s create two factory function’s one which creates “Address” object and the other which creates “Phone” objects.

```
function CreateAddress()
{
        var add = new Address();
        return add;
}
function CreatePhone()
{
        var phone =  new Phone();
        return phone;
}
```

Now let’s create a main factory function which uses the above two small factory functions and gives us all the necessary permutation and combination.
In the below factory you can see we have three functions:-

- “CreateWithAddress” which creates “Customer” with “Address” objects inside it.
- “CreateWithPhone” which creates “Customer” object with “Phone” objects inside it.

- “CreateWithPhoneAddress” which creates “Customer” object with aggregated “Phone” and “Address” objects.

```
function CreateCustomer() {


        return {
            CreateWithAddress: function () {
                var cust = new Customer();
                cust.Address = CreateAddress();
                return cust;
            },
            CreateWithPhone: function () {
                var cust = new Customer();
                cust.Phone = {};
                cust.Phone = CreatePhone();
                return cust;
            }
            ,
            CreateWithPhoneAddress: function () {
                debugger;
                var cust = new Customer();
                cust.Phone = CreatePhone();
                cust.Address = CreateAddress();
                return cust;
            }
        }
    }
```

Below is a simple “CustomerController” which takes “CustomerFactory” as the input. Depending on “TypeOfCustomer” it creates with “Address” , “Phones” or both of them.

```
function CustomerController($scope, Customerfactory)
    {


        $scope.Customer = {};
        $scope.Init = function(TypeofCustomer)
```

```
        {


            if (TypeofCustomer == "1")
            {
                $scope.Customer = Customerfactory.CreateWithAddress();
            }
            if (TypeofCustomer ==  "2")
            {
                $scope.Customer = Customerfactory.CreateWithPhone();
            }
            if (TypeofCustomer == "3") {
                $scope.Customer = Customerfactory.CreateWithPhoneAddress();
            }
        }
    }
```

You also need to tell Angular that the “CreateCustomer” method needs to be passed in the input. For that we need to call the “Factory” method and map the “CreateCustomer” method with the input parameter “CustomerFactory” for dependency injection.

```
var app = angular.module("myApp", []); // creating a APP
app.controller("CustomerController", CustomerController); // Register the VM
app.factory("Customerfactory", CreateCustomer);
```

So if we consume the “CustomerController” in UI , depending on situation it creates different flavors of “Customer” object. You can in the below code we have three different “DIV” tags and depending on the “TypeofCustomer” we are displaying data.

My other interview question articles

Jquery, JSON and Less Interview questions with answers

http://www.codeproject.com/Articles/778374/JQUERY-JSON-Angular-and-Less-Interview-questions

100 important ASP.NET MVC interview questions

http://www.codeproject.com/Articles/556995/ASP-NET-MVC-interview-questions-with-answers

HTML 5 Interview questions with answers

http://www.codeproject.com/Articles/702051/important-HTML-Interview-questions-with-answe

WPF interview questions with answers

http://www.codeproject.com/Articles/744082/WPF-Interview-questions-with-answers