Session 7

Building Single Page Applications (SPAs) in AngularJS

Session Objectives

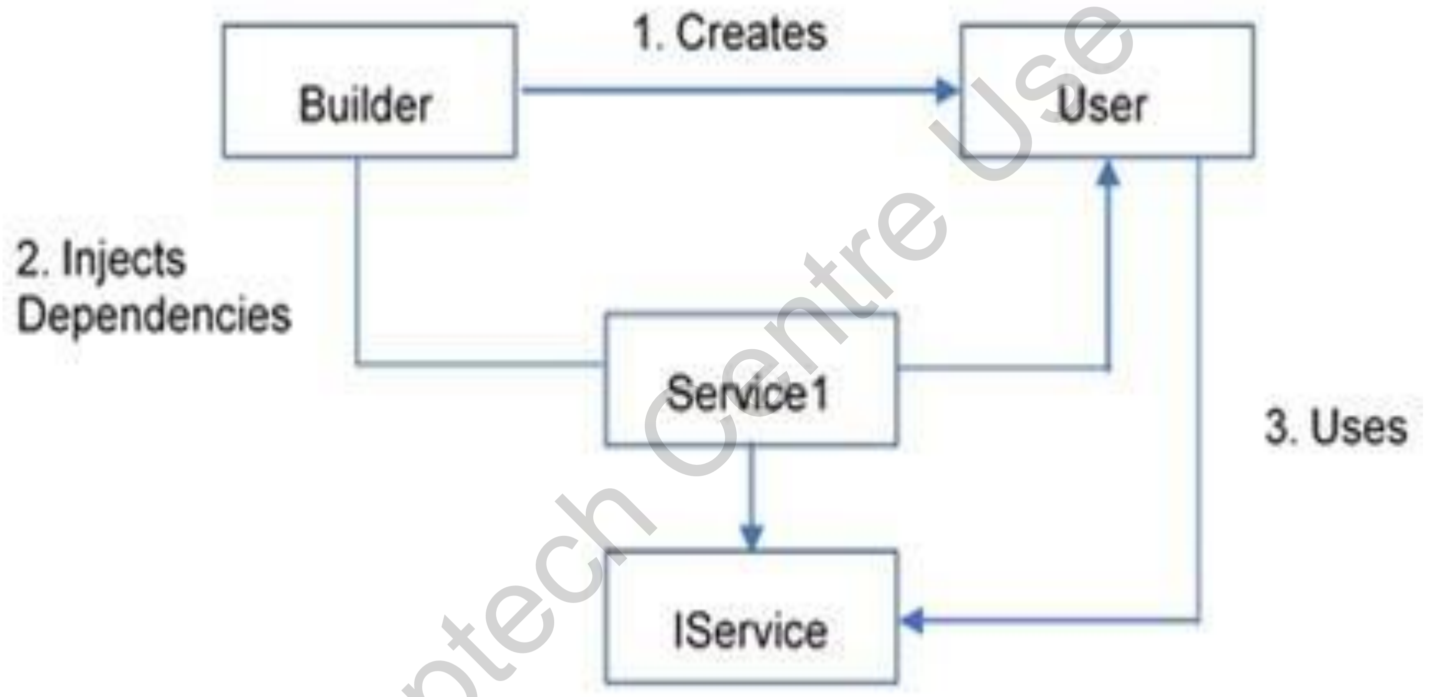
- ✓ Explain dependency injection and its working in AngularJS
- ✓ Describe factory and service in AngularJS
- ✓ Outline the differences between factory and service and their uses
- ✓ Explain the usage of SPAs in AngularJS

Dependency Injection

- Is a **technique** for passing a dependent object into another object to make all functionality of the former available to the latter.
- Prevents a component from directly referring to another component, but allows obtaining a reference to it.
- Eliminates hard-coded dependencies by requesting a dependent functionality instead of creating it by coding.
- Makes AngularJS applications maintainable.
- Modularizes an application by splitting it into several components, all of which are injectable into each other.

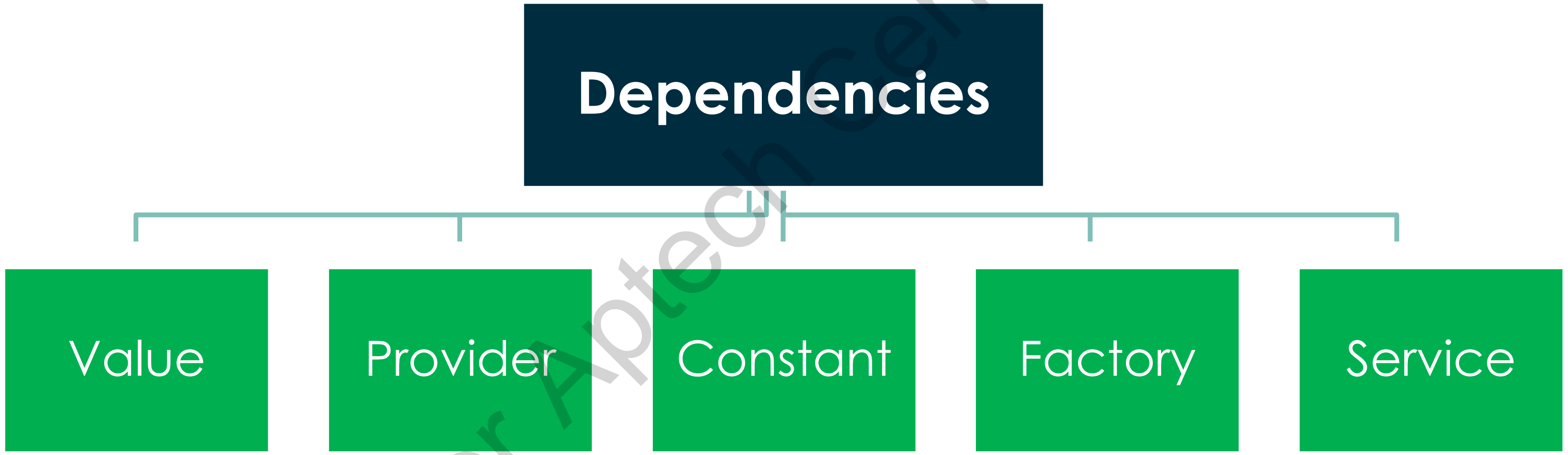
Dependency Injection

2-2



Objects and Components as Dependencies

In AngularJS, five objects or components exist that can inject into each other as dependencies:



Value

A value:

- Refers to a JavaScript object, string, or a number that injects into a controller during the config phase.
- Injects into a factory, service, or a controller.

Injecting a value into a module

```
var newMod =
angular.module("newMod",
[]);
newMod.value("number",
10);
newMod.value("string",
"employee");
newMod.value("object", {
value1 : 50, value2 :
"manager"} );
```

Injecting a value into a controller

```
<html ng-app="app">
 <head>
 <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js">
</script>
 <script>
   var app = angular.module("app", [ ]);
   app.value('empId', '101');
   app.controller("MyController", ['$scope', "empId", function
   MyController($scope, empId){
        $scope.empId = empId;
       }
    ]);
</script>
</head>
<body ng-controller='MyController'>
<p>Hello Employee, <b>{{empId}}</b>! Welcome.</p>
</body>
</html>
```

Provider

A provider:

- Creates a service or factory during the config phase using the `$provide` service.
- Is a distinct factory method with the `$get()` method.
- Returns a value, factory, or a service.

AngularJS allows defining a provider through the **`provider()`** method, which in turn, invokes the **`$provide`** service.

A developer can also specify functions using `config()` and `run()`, which then run at configuration and run time, respectively. These functions are injectable with dependencies.

Constant

A constant:

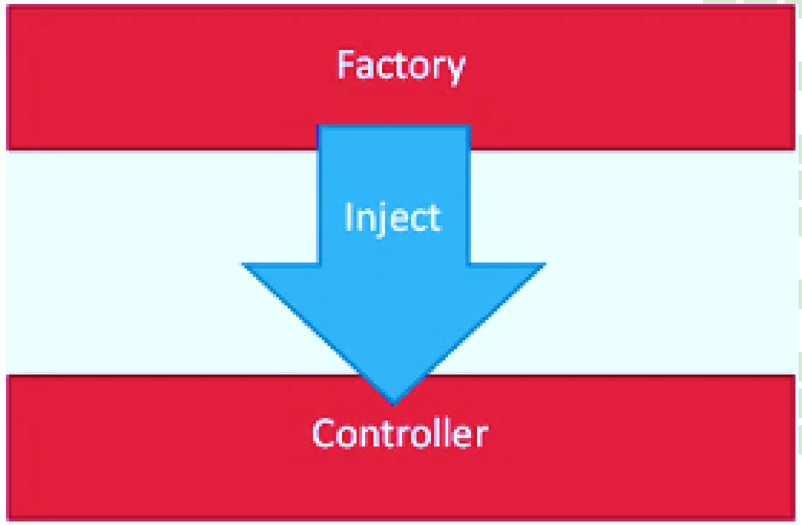
- Remains **fixed** throughout execution.
- Is injectable into config(), controller, and a provider.

Example shows how to inject a constant into a controller

```
<html>
   <head>
      <title>Angular JS Services</title>
      <script src =
    "https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js">
     </script>
   </head>
   <body>
     <div ng-app = "mainApp" ng-controller = "myController">
        <p>Enter radius of circle: <input type = "number" ng-model =
        "number" /></p>
         <button id="btnCal" ng-click="area()">Area</sup></button>
          <p>{{pi}}</p>
                        <p>Result: {{result}}</p>
      </div>
      <script>
       var mainApp = angular.module("mainApp", []);
       mainApp.constant("pi", "3.14");
       mainApp.controller('myController', function($scope, pi) {
          $scope.area=function(a) {
          $scope.result = pi*($scope.number * $scope.number);
          };
       });
      </script>
   </body>
</html>
```

Factory

- Is a component that is technically a function and is injectable with values.
- Returns a re-usable value when a service or a controller requires it.
- Implements the `factory()` method for creating and returning a value.
- Is injectable into a controller.



Creating a Factory

1-2

Following example shows how to create a service using the `factory()` function and inject it into a controller.

```
<html>
 <head>
   <title>Angular JS Services</title>
   <script src =
  "https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js">
   </script>
 </head>
 <body>
     <div ng-app = "mainApp" ng-controller = "DemoController">
     <p>Enter a number: <input type = "number" ng-model = "number" /></p>
         <button ng-click = "square()">Square</button>
         <p>Result: {{result}}</p>
     </div>
     <script>
       var mainApp = angular.module("mainApp", []);
       mainApp.factory('MathService', function() {
               // Define a factory
                    var factory = {};
             //Assign a function to it
               factory.square = function(a) {
                         return a * a;
```

Creating a Factory

```
                  }
                 return factory;
                 });
           //Inject the service created using factory into the controller
         mainApp.controller('DemoController', function($scope,
         MathService) {
                       $scope.square = function() {
                          $scope.result =
                        MathService.square($scope.number);
                       }
                    });
        </script>
     </body>
</html>
```

Service

A service:

- Is a single JavaScript object holding a set of functions that are injectable into a controller.
- Is used to create a service that returns no value.
- Implements the service() constructor function for creating a service object and adding functions and properties to it.
- Is injectable into a controller, filter, directive, or another service.

Creating a Service

Following example shows how to invoke an already existing function, `MyService`, by using `service()`.

```
<html>
   <head>
      <title>Angular JS Services</title>
      <script src =
"https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
      </script>
   </head>
   <body>
        <div ng-app = "mainApp" ng-controller = "CalcController">
         <p>Enter a number: <input type = "number" ng-model = "number" /></p>
         <button ng-click = "square()">Square</sup></button>
         <p>Result: {{result}}</p>
      </div>
                      <script>
         var mainApp = angular.module("mainApp", []);
         mainApp.service('MyService', function() {
            this.square = function(a) {
               return a * a;
            }
```

Creating a Service

2-2

```
        });
        //Inject the created service into the controller
           mainApp.controller('CalcController', function($scope, MyService) {
              $scope.square = function() {
                 $scope.result = MyService.square($scope.number);
              }
           });
        </script>
     </body>
</html>
```

Creating a Factory and a Service 1-2

Following example shows how to create a factory and a service, both returning Hello string:

```
<html>
<head>
 <title>Angular JS Services</title>
 <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
 </script>
</head>
   <body>
      <div ng-app = "mainApp" ng-controller = "DemoController">
         <p>Enter a number: <input type = "number" ng-model = "number" /></p>
         <button ng-click = "cube1()">X<sup>3</sup></button>
         <p>Result (Using Factory): {{result1}}</p>
         <button ng-click = "cube2()">X<sup>3</sup></button>
         <p>Result (Using Service): {{result2}}</p>
      </div>
      <script>
         var mainApp = angular.module("mainApp", []);
         mainApp.factory('Math', function() {
            var factoryObj = {};
            factoryObj.multiply = function(a) {
               return a * a* a;
            }
            return factoryObj;
         });
```

Creating a Factory and a Service 2-2

```
      mainApp.service('CalcService', function() {
            this.cube = function(a) {
               return a*a*a;
            }
         });
      mainApp.controller('DemoController', function($scope, CalcService,
Math) {
            // Using Service
            $scope.cube1 = function() {
               $scope.result1 = CalcService.cube($scope.number);
            }
         // Using Factory
           $scope.cube2 = function() {
               $scope.result2 = Math.multiply($scope.number);
            }
         });
     </script>
   </body>
</html>
```

Factory versus Service

Point of Distinction	Factory	Service
Function Type	Is a function that returns an object or a value.	Is a constructor function that uses the new keyword to declare an object. It is instantiated only when a component depends on it.
Use	Is used for non-configurable services. It can also be used as a service for replacing complex logic. Go for it if you are using an object.	Is used for inserting simple logic. Go for it if you are using a class.
Properties	Are defined without this keyword.	Are defined with this keyword.
Friendly Injections	Are not supported.	Are supported.
Primitives	Are created.	Are not created.
Preferable Choice	Is more preferable due to its class-like definition.	Is preferred only for defining utility services or using ES6 classes.

AngularJS Dynamic Templates

A dynamic template:

- Allows adding services in the desired order or dynamically.
- Is made by implementing a custom directive for each service.
- Consists of custom directives that extend the HTML functionality and are associated with elements, attributes, styles, and comments.
- Works at the time of loading by invoking the `compile()` method of the directive once and processing via the directive's `link()` method.

Creating a Dynamic Template 1-2

Following example shows how to define a custom directive for a dynamic template:

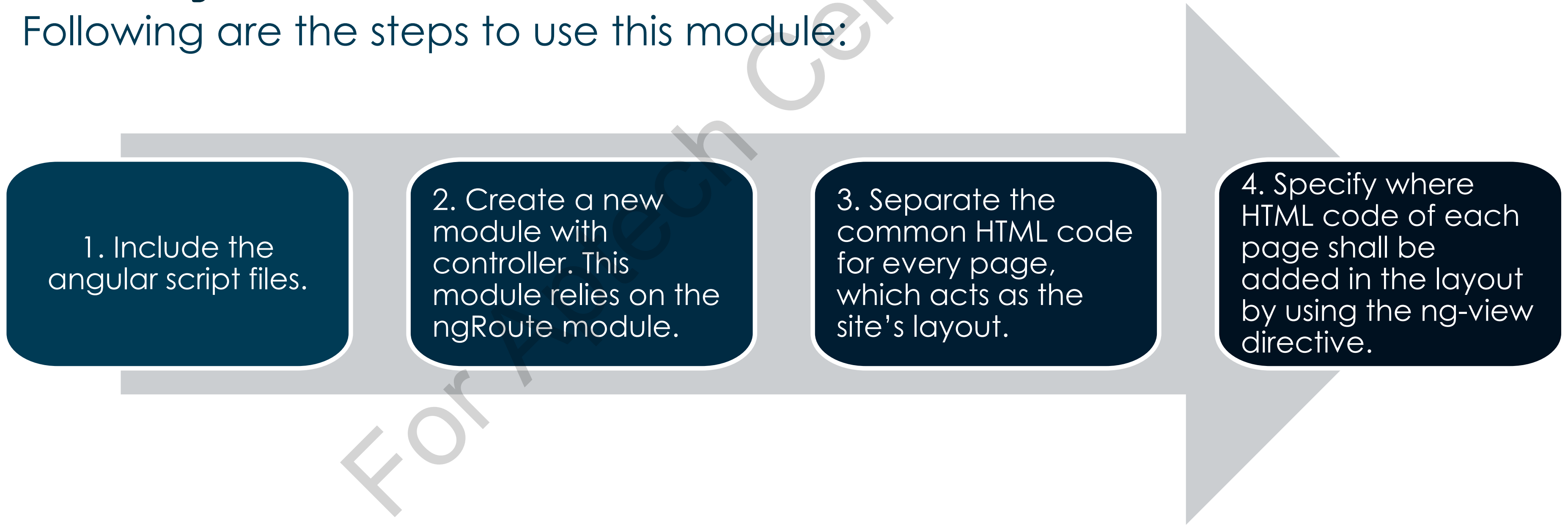
```
<!DOCTYPE html>
<html ng-app="dynamictemp">
  <head>
   <meta charset="utf-8" />
   <title>AngularJS Dynamic Template</title>
   <script>document.write('<base href="' + document.location + '" />');
   </script>
    <script src = "https://ajax.googleapis.com/ajax/libs/angularjs/1.7.9/angular.min.js">
   </script>
    <script src="app.js"></script>
  </head>
  <body ng-controller="MainCtrl">
    <select ng-model="model.loanType">
      <option value="">Select Loan Type</option>
      <option value="1">Personal Loan</option>
      <option value="2">Housing Loan</option>
    </select>
      <loan-detail-form loan="model"/>
      <br/>
      <br/>
    <script type="text/ng-template" id="template1">
      <form>
```

Creating a Dynamic Template 2-2

```
<br/>
<br/>
<fieldset>
          <legend>Personal Loan</legend>
          <label>Document needed:</label>
          <input type="text" ng-model="loan.attributeA">
        </fieldset>
     </form>
    </script>
 <!--Design the look as an enclosing box with elements inside-->
    <script type="text/ng-template" id="template2">
      <form>
      <br/>
      <br/>
        <fieldset>
          <legend>Housing Loan</legend>
          <label>Document needed:</label>
          <input type="text" ng-model="loan.attributeB" readonly><br/>
          <label>Document needed:</label>
          <input type="text" ng-model="loan.attributeC" readonly>
        </fieldset>
      </form>
    </script>
  </body>
</html>
```

Steps for Building an SPA

- Each AngularJS application is begun by designing a module.
- A module is a container for holding different components such as controllers and services.
- Next, you specify the name of the module as well as controller as the value of **`ng-app`** and **`ng-controller`** attributes, respectively.
- Next, you utilize the routing capabilities of AngularJS to make an SPA by using the built-in **`ngRoute`** module.
- Following are the steps to use this module:



1. Include the angular script files.

2. Create a new module with controller. This module relies on the ngRoute module.

3. Separate the common HTML code for every page, which acts as the site's layout.

4. Specify where HTML code of each page shall be added in the layout by using the ng-view directive.

Summary

- Dependency injection is a pattern or technique for adding a dependent functionality into a module at the time of execution without coding for it.
- The benefits of dependency injection include no hard-coded dependencies, modularized applications, easy configurations and code changes, reusable modules, and mock testing of applications.
- AngularJS allow injecting values, providers, constants, factories, and services into each other as dependencies.
- Values are injected into a factory, controller, or a service.
- All the different ways of creating a service in AngularJS ultimately use $provide.
- The config() function accepts only a provider of a service or a constant as a parameter.
- A factory uses a function that returns a value, while a service is a constructor function, which uses the new keyword for creating an object and adding functions and properties to it.
- Factories and services are providers.
- Creating a dynamic template involves using custom directives per service such that the services are added in a random order.