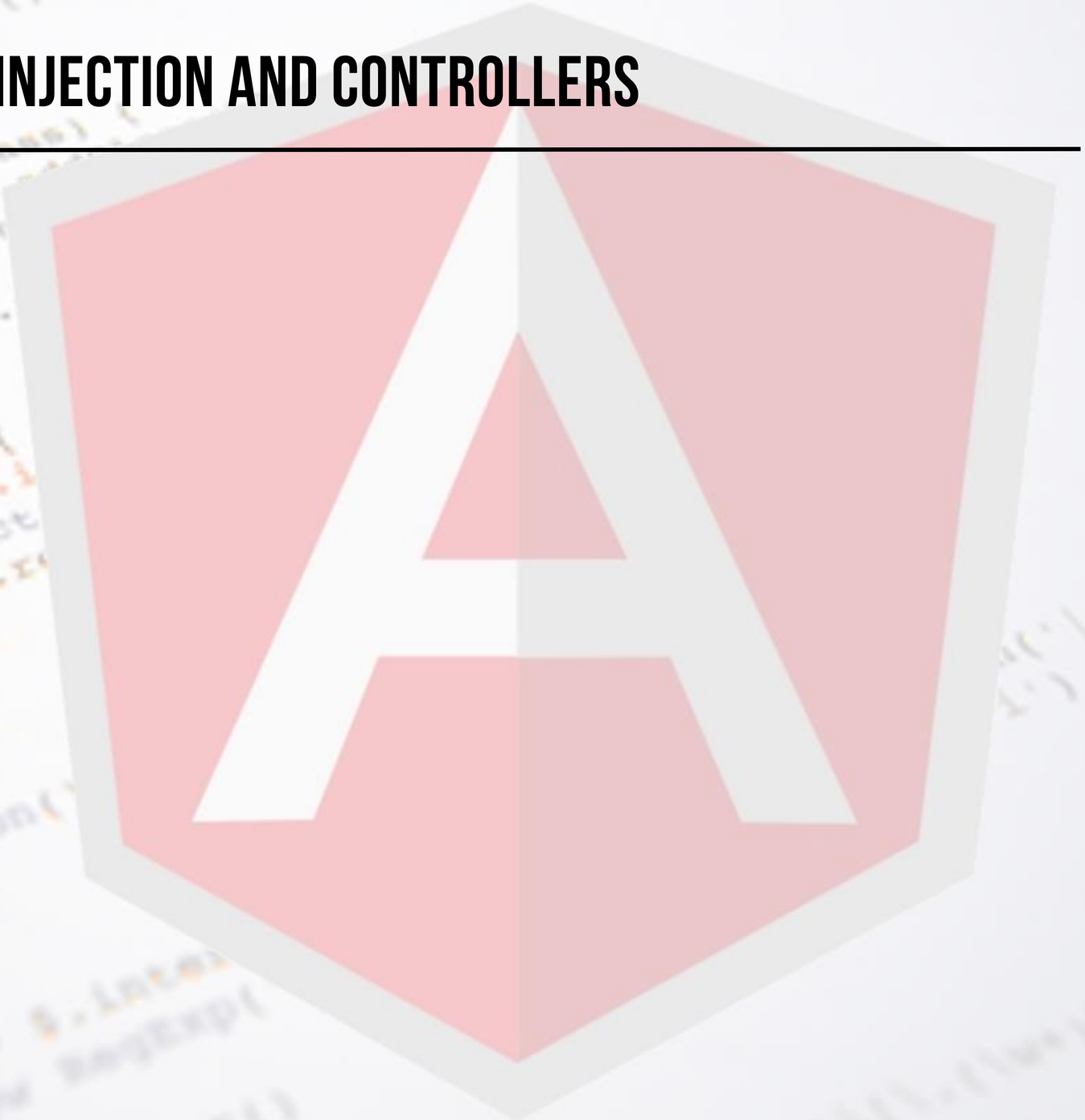
ANGULAR JS

A SINGLE PAGE APPLICATION FRAMEWORK

DEPENDENCY INJECTION AND CONTROLLERS

DAY 1

SESSION 2



DEPENDENCY INJECTION AND CONTROLLERS

DAY 1, SESSION - 2

1 | MODULE

2 | CREATE CONTROLLER

3 | EXPRESSION AND CONTROL FLOW LOGIC

4 | SHARING DATA BETWEEN CONTROLLERS

5 | SCOPE INHERITANCE

6 | DEPENDENCY INJECTION

1. MODULE

A module is a collection of services, directives, controllers, filters, and configuration information.

The proper way to create an angular module is to start with angular.module and pass app name and its dependency array, if for now we don't have dependency just pass the blank array.
Here we have created an angular module with name eShop with no dependency.

```
angular.module("eShop", []);
```

To add property to module either we can use chaining when we write properties like controller, service, and directive in the same file or get a reference by angular.module with first

```
angular.module("eShop", []).controller();
```

```
angular.module("eShop").controller();
```

1. SCOPE

Scope is an object that refers to the application model. It is an execution context for expressions. Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch expressions and propagate events.

Scope characteristics

- Scopes provide APIs ($watch) to observe model mutations.
- Scopes provide APIs ($apply) to propagate any model changes through the system into the view from outside controllers, services, Angular event handlers.
- Scopes can be nested to limit access to the properties of application components while providing access to shared model properties. Nested scopes are either "child scopes" or "isolate scopes".
- A "child scope" (prototypically) inherits properties from its parent scope.
- An "isolate scope" does not.
- Scopes provide context against which expressions are evaluated.

2. CONTROLLER

Controller is a JavaScript constructor function that operates the Angular Scope.

How controller works:
When a Controller is attached to the DOM via the **ng-controller** directive, Angular will instantiate a new Controller object, using the specified Controller's constructor function. A new child scope will be created and made available as an injectable parameter to the Controller's constructor function as **$scope**.

How to write controller:
The following example demonstrates creating a MainController, which attaches a `shopName` property containing the string `'Angular E-Store'` to the $scope:

```
eShop.controller('MainController', ['$scope', function($scope) {
  $scope.shopName = 'Angular E-Store';
}]);
```

2. CONTROLLER……

Use controllers to:

- Set up the initial state of the $scope object.
- Add behavior to the $scope object.

Do not use controllers to:

- Manipulate DOM — Controllers should contain only business logic. Putting any presentation logic into Controllers significantly affects its testability. Angular has data-binding for most cases and directives to encapsulate manual DOM manipulation.
- Format input — Use angular form controls instead.
- Filter output — Use angular filters instead.
- Share code or state across controllers — Use angular services instead.
- Manage the life-cycle of other components (for example, to create service instances).

3. EXPRESSION AND CONTROL LOGIC

Expressions : Angular expressions are code block enclosed with double curly braces {{ }} used in HTML to bind model data which its controller scope holds. Expression also may have primitive data operations like string concatenation, numerical operations and so on.

For example, these are valid expressions in Angular:

```
{{1+2}}
{{a+b}}
{{user.name}}
{{items[index]}}
```

Angular Expressions vs. JavaScript Expressions

- Evaluated against a scope object.
- Forgiving to undefined and null.
- No Control Flow Statements
- No Function Declarations
- No RegExp
- No Comma
- Filters allowed

3. EXPRESSION AND CONTROL LOGIC.....

Few list of mostly used inbuilt directives.

Name	Description
ng-disabled="expression"	disables a given control.
ng-show="expression"	shows a given control.
ng-hide="expression"	hides a given control.
ng-click="expression"	represents a AngularJS click event.
ng-if="expression"	If expression returns true it render html on which directive applied
ng-repeat="key in object"	Iterate through array

4. SHARING DATA BETWEEN CONTROLLERS

Angular Service/Factory is a pluggable component that wired together using Dependency Injection.

Angular services are:

- **Lazily instantiated** – Angular only instantiates a service when an application component depends on it.

- **Singletons** – Each component dependent on a service gets a reference to the single instance generated by the service factory.
- Angular offers several useful services (like $http), but it provides you flexibility so that you can also create your own.

Ways to share data ?
- value
- constant
- factory
- service
- provider

Value
Syntax: `module.value("defaultInput", 5);`
Result: We use

4. SHARING DATA BETWEEN CONTROLLERS

Services
Syntax: `module.service( 'serviceName', function );`
Result: When declaring serviceName as an injectable argument you will be provided with an instance of the function. In other words new FunctionYouPassedToService().

Factories
Syntax: `module.factory( 'factoryName', function );`
Result: When declaring factoryName as an injectable argument you will be provided with the value that is returned by invoking the function reference passed to module.factory.

Providers
Syntax: `module.provider( 'providerName', function );`
Result: When declaring providerName as an injectable argument you will be provided with ProviderFunction().$get(). The constructor function is instantiated before the $get method is called - ProviderFunction is the function reference passed to module.provider.

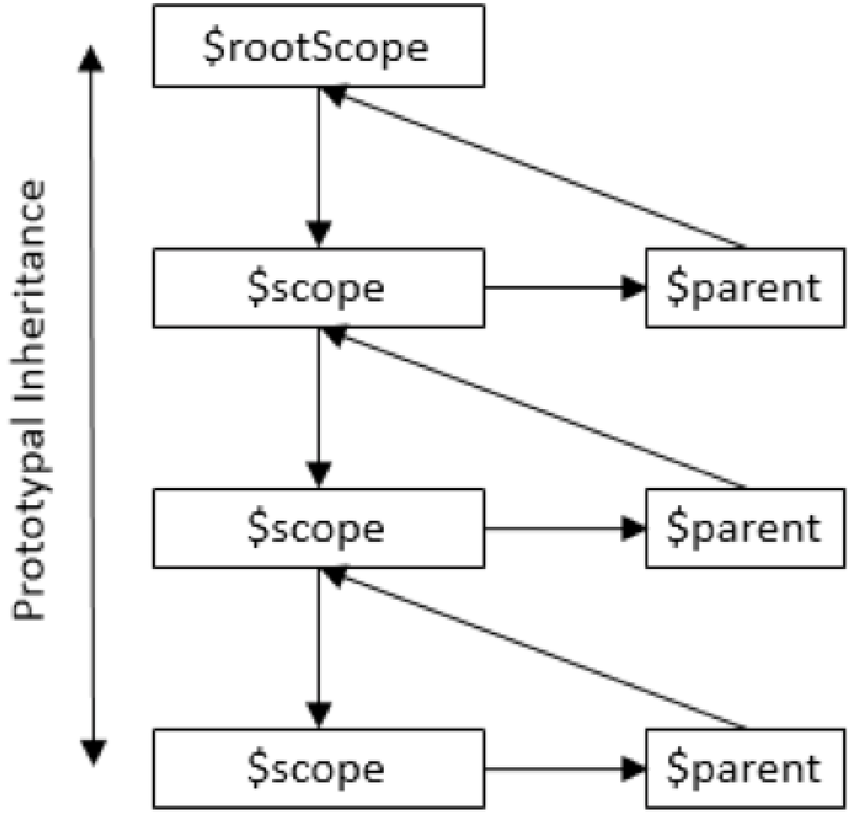
Providers have the advantage that they can be configured during the module configuration phase.

```
var myApp = angular.module('myApp', []);
//Service style, probably the simplest one
myApp.service('helloWorldFromService', function() { this.sayHello = function() { return "Hello,
World!" }; });

//Factory style, more involved but more sophisticated myApp.factory('helloWorldFromFactory',
function() { return { sayHello: function() { return "Hello, World!" } }; });

//Provider style, configurable version
myApp.provider('helloWorld', function() {
// In the provider function, you cannot inject any
// service or factory. This can only be done at the "$get" method.
this.name = 'Default';
this.$get = function() { var name = this.name;
        return { sayHello: function() {
                                return "Hello, " + name + "!" }
                } };
this.setName = function(name) { this.name = name; }; });

//Hey, we can configure a provider!
myApp.config(function(helloWorldProvider){
```

4. SCOPE CHAIN

Scope Inheritence

- Each $scope is an instance of Scope class.
- When Angular app bootstrapped $rootScope instance gets created.
- When ng-controller directive attached with dom usin Scope.$new() method a new scope is created as $rootScope child.

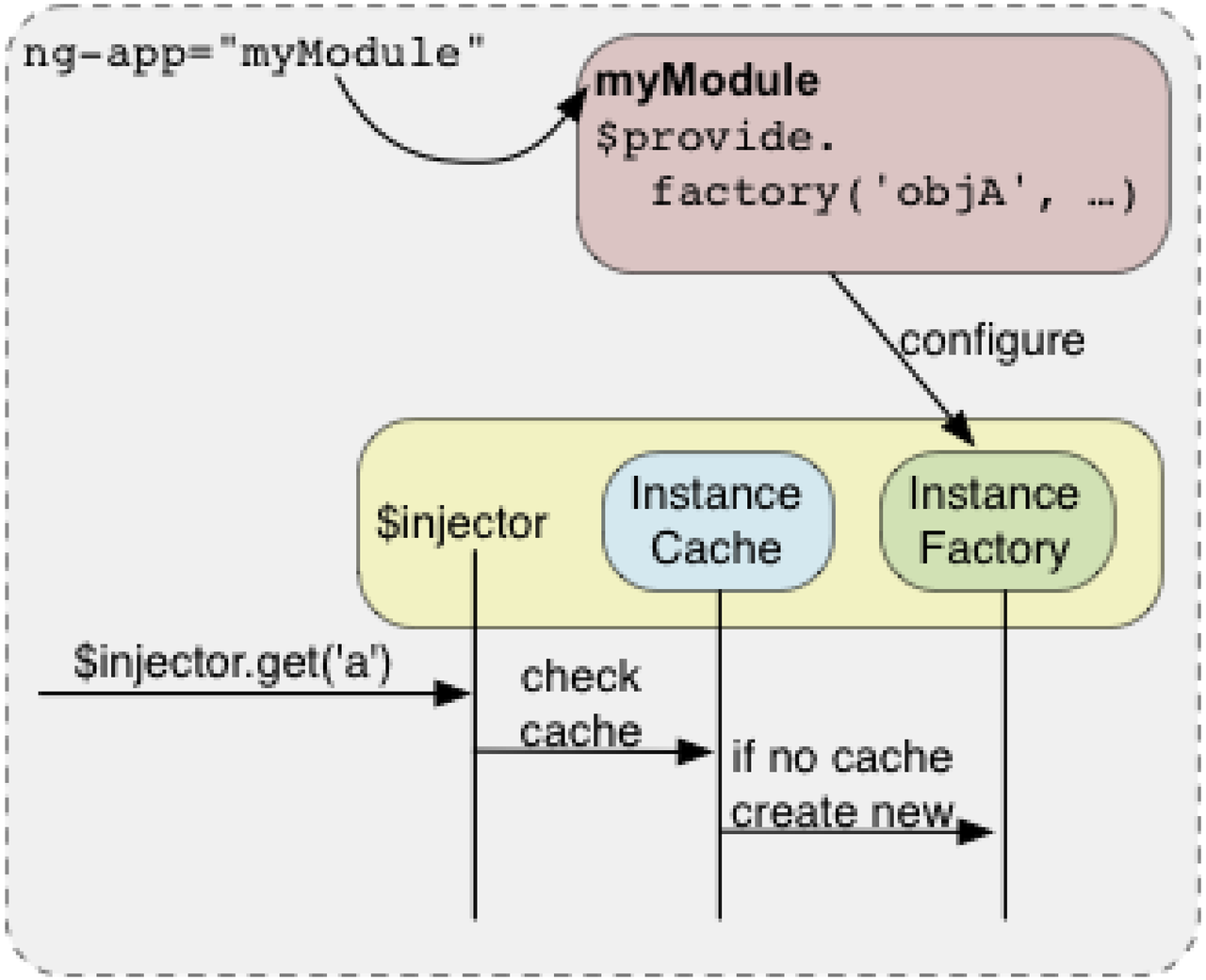


4. DEPENDENCY INJECTION

Dependency Injection (DI) is a software design pattern that deals with how components get hold of their dependencies.

AngularJS provides a supreme Dependency Injection mechanism. It provides following core components which can be injected into each other as dependencies.

- value
- factory
- service
- provider
- constant



ANY QUESTIONS?