# **Understanding Angular in Java**

Note: This is an exercise to simplify understanding of Angular. Definitions are explained in Java context.

**Module:**

Is similar to a Java class. It is defined using syntax.

Module will have controllers and services that perform business logic.

angular.module (“myNewModule”) {

}

Similar to java inner classes, we can have 3 flavors of Angular

* One Java Script file can have multiple modules.
* One module can span across multiple Java script files.
* One Module One JS File **(Recommended, Advised….You can call it whatever but follow this).**

**Controller:**

Controllers are like Java function, they are typically designed to handle business logic.

They can be named inside module or outside a module in which case it is global.

Declaring global controllers is not recommended.

**Model:**

This is similar to form bean java object that is injected to JSP’s.

These are the variables that you create in ui.

This is where Javascript, JQuery and Angular Diverge in essence. Ng-model is a directive supported by angular, which holds form value and is in a continuous relationship with backend. Which means any changes to model are reciprocated instantly to UI fields and form fields. They is no need for form submit or ajax call to bind values. Angular handles is automatically, and “**Balieve me**” user does even feel a blink of screen. It is **one 100 Percent** seamless.

**Directives**

* $scope.$watch('funding.startingEstimate', computeNeeded); //watch the expression 'funding.startingEstimate and call computeNeeded() when it changes' - note the expression to watch is in quotes
* event-handling directives include ng-click
* ng-show and ng-hide function by setting the CSS element styles to display:block and display:none

**$watch()**

The $watch() function is possibly the most used scope function: $watch(watchFunction, watchAction, deepWatch)

* *watchFunction* is an Angular expression (passed in as a 'string') or function that is being watched for changes
* *watchAction* is an Angular expression or function that is carried out when the *watchFunction* changes
* If *deepWatch* is set to true it examines each object within the *watchFunction* (so if the watchFunction is an array, it walks all the properties for the array)

Using the dreg(); function after the $watch() function de-registers the event listener

The most performance-efficient way of using $watch() can sometimes be to set up a watch run every time Angular evaluates the page (avoiding expensive data binding where certain functions are run multiple times)

To **watch multiple items** you can either put them in an array (and pass in deepWatch as true in your $watch() function) or pass in a concatenated set of properties, for example to watch all the values in the things array $scope.$watch('things', callMe(...), true); (setting deepWatch to true then means that any changes to any items in the *things* array will trigger the *callMe()* function)

### Data filters

Data filters allow you to define how to transform your data from within the template - the key thought process here is that you would do this within your view (instead of your controller or model) because it's for things that are only important when displaying the data to humans but make no difference to the logic in your controller (like adding a dollar sign for currency for example). Format: {{angularExpression | filterName : parameter1 : parameter2 ... | filterName2 ...}}

**Services vs Controller’s**

**Controllers** are typically used to be bound with a view. Controllers manage a view's life cycle, and should be thought of as View Controllers. A new controller will be created for each instance of a view, meaning that if you navigate away from a certain view, and then back again - or if you have more than once instance of a certain view, a new controller will be created each time.

**Services** are typically used as the business logic of your application. Services are similar to singletons in the sense that they are created once, and the instance is maintained throughout the entire life cycle of your application. It is a good place to put your logical functions which many views or components will require, and also hold global cache which needs to be accessed throughout multiple areas in your application.

angular.module('finance3', [])

.factory

**2-Way data binding in Angular**

Angular templates work differently. First the template (which is the uncompiled HTML along with any additional markup or directives) is compiled on the browser. The compilation step produces a live view. Any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view. The model is the single-source-of-truth for the application state, greatly simplifying the programming model for the developer. You can think of the view as simply an instant projection of your model.

Because the view is just a projection of the model, the controller is completely separated from the view and unaware of it. This makes testing a snap because it is easy to test your controller in isolation without the view and the related DOM/browser dependency.

# **Services**

Angular services are substitutable objects that are wired together using [dependency injection (DI)](https://docs.angularjs.org/guide/di). You can use services to organize and share code across your app.

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 [$http](https://docs.angularjs.org/api/ng/service/$http)), but for most applications you'll also want to [create your own](https://docs.angularjs.org/guide/services#creating-services).

## **Creating Services**

Application developers are free to define their own services by registering the service's name and **service factory function**, with an Angular module.

The **service factory function** generates the single object or function that represents the service to the rest of the application. The object or function returned by the service is injected into any component (controller, service, filter or directive) that specifies a dependency on the service.

# **What are Scopes?**

[Scope](https://docs.angularjs.org/api/ng/type/$rootScope.Scope) is an object that refers to the application model. It is an execution context for [expressions](https://docs.angularjs.org/guide/expression). Scopes are arranged in hierarchical structure which mimic the DOM structure of the application. Scopes can watch [expressions](https://docs.angularjs.org/guide/expression) and propagate events.

## **Scope Life Cycle**

The normal flow of a browser receiving an event is that it executes a corresponding JavaScript callback. Once the callback completes the browser re-renders the DOM and returns to waiting for more events.

When the browser calls into JavaScript the code executes outside the Angular execution context, which means that Angular is unaware of model modifications. To properly process model modifications the execution has to enter the Angular execution context using the [$apply](https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$apply)method. Only model modifications which execute inside the $apply method will be properly accounted for by Angular. For example if a directive listens on DOM events, such as [ng-click](https://docs.angularjs.org/api/ng/directive/ngClick) it must evaluate the expression inside the $apply method.

After evaluating the expression, the $apply method performs a [$digest](https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$digest). In the $digest phase the scope examines all of the $watchexpressions and compares them with the previous value. This dirty checking is done asynchronously. This means that assignment such as $scope.username="angular" will not immediately cause a $watch to be notified, instead the $watch notification is delayed until the $digest phase. This delay is desirable, since it coalesces multiple model updates into one $watch notification as well as guarantees that during the $watch notification no other $watches are running. If a $watch changes the value of the model, it will force additional$digest cycle.

1. **Creation**

The [root scope](https://docs.angularjs.org/api/ng/service/$rootScope) is created during the application bootstrap by the [$injector](https://docs.angularjs.org/api/auto/service/$injector). During template linking, some directives create new child scopes.

1. **Watcher registration**

During template linking, directives register [watches](https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$watch) on the scope. These watches will be used to propagate model values to the DOM.

1. **Model mutation**

For mutations to be properly observed, you should make them only within the [scope.$apply()](https://docs.angularjs.org/api/ng/type/$rootScope.Scope" \l "$apply). Angular APIs do this implicitly, so no extra $apply call is needed when doing synchronous work in controllers, or asynchronous work with [$http](https://docs.angularjs.org/api/ng/service/$http), [$timeout](https://docs.angularjs.org/api/ng/service/$timeout) or [$interval](https://docs.angularjs.org/api/ng/service/$interval)services.

1. **Mutation observation**

At the end of $apply, Angular performs a [$digest](https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$digest) cycle on the root scope, which then propagates throughout all child scopes. During the $digest cycle, all $watched expressions or functions are checked for model mutation and if a mutation is detected, the $watch listener is called.

1. **Scope destruction**

When child scopes are no longer needed, it is the responsibility of the child scope creator to destroy them via [scope.$destroy()](https://docs.angularjs.org/api/ng/type/$rootScope.Scope" \l "$destroy) API. This will stop propagation of $digest calls into the child scope and allow for memory used by the child scope models to be reclaimed by the garbage collector.

### Scopes and Directives

During the compilation phase, the [compiler](https://docs.angularjs.org/guide/compiler) matches [directives](https://docs.angularjs.org/api/ng/provider/$compileProvider#directive) against the DOM template. The directives usually fall into one of two categories:

* Observing [directives](https://docs.angularjs.org/api/ng/provider/$compileProvider#directive), such as double-curly expressions {{expression}}, register listeners using the [$watch()](https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$watch) method. This type of directive needs to be notified whenever the expression changes so that it can update the view.
* Listener directives, such as [ng-click](https://docs.angularjs.org/api/ng/directive/ngClick), register a listener with the DOM. When the DOM listener fires, the directive executes the associated expression and updates the view using the [$apply()](https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$apply) method.

When an external event (such as a user action, timer or XHR) is received, the associated [expression](https://docs.angularjs.org/guide/expression) must be applied to the scope through the [$apply()](https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$apply) method so that all listeners are updated correctly.

### Directives that Create Scopes

In most cases, [directives](https://docs.angularjs.org/api/ng/provider/$compileProvider#directive) and scopes interact but do not create new instances of scope. However, some directives, such as [ng-controller](https://docs.angularjs.org/api/ng/directive/ngController)and [ng-repeat](https://docs.angularjs.org/api/ng/directive/ngRepeat), create new child scopes and attach the child scope to the corresponding DOM element. You can retrieve a scope for any DOM element by using an angular.element(aDomElement).scope() method call. See the [directives guide](https://docs.angularjs.org/guide/directive#isolating-the-scope-of-a-directive) for more information about isolate scopes.

### Controllers and Scopes

Scopes and controllers interact with each other in the following situations:

* Controllers use scopes to expose controller methods to templates (see [ng-controller](https://docs.angularjs.org/api/ng/directive/ngController)).
* Controllers define methods (behavior) that can mutate the model (properties on the scope).
* Controllers may register [watches](https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$watch) on the model. These watches execute immediately after the controller behavior executes.

See the [ng-controller](https://docs.angularjs.org/api/ng/directive/ngController) for more information.

### Scope $watch Performance Considerations

Dirty checking the scope for property changes is a common operation in Angular and for this reason the dirty checking function must be efficient. Care should be taken that the dirty checking function does not do any DOM access, as DOM access is orders of magnitude slower than property access on JavaScript object.