Angular

Course Objective

- To understand and develop rich interactive Web Application using Angular.

Session Plan

- Introduction to Angular
- Typescript
- Angular Architecture
  - Module
  - Component
  - Template
  - Metadata
  - Data Binding
  - Directives
  - Services
  - Dependency Injection
- Routers
- Forms
- Using a Service

Introduction to Angular

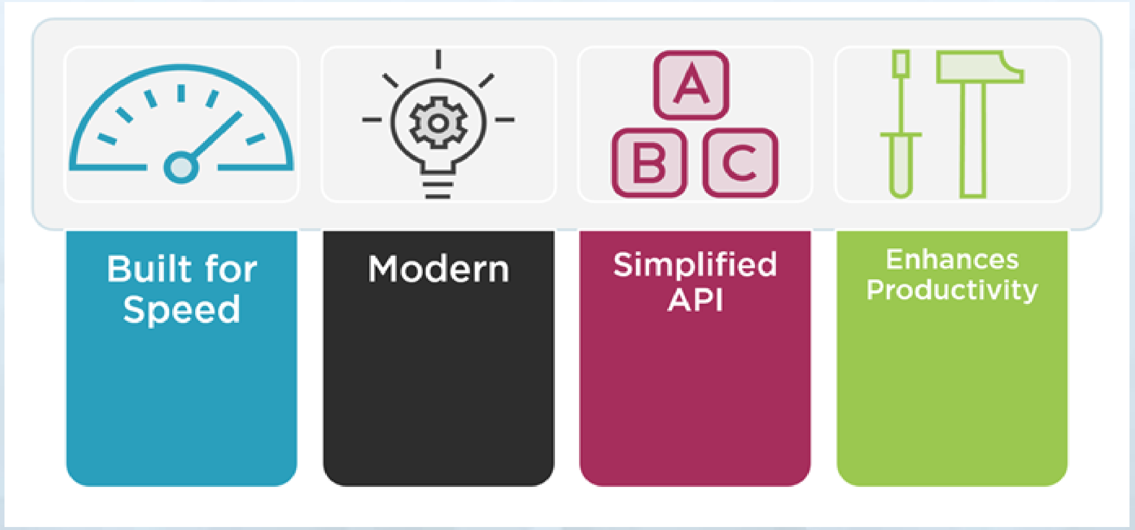
- Angular is a platform and framework for building client applications in HTML and TypeScript.
- Angular is itself written in TypeScript.
- It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.

Introduction to Angular

- Angular is the most advanced framework for the web.
- It empowers developers to build applications that live on the web, mobile, or the desktop.
- It aims to simplify both the development and testing of such applications by providing a framework for client-side model–view–controller (MVC) and model–view–view-model (MVVM) architectures, along with components commonly used in rich internet applications.
- Angular has rebuilt the entire framework in TypeScript,
- Angular combines declarative templates, dependency injection, end to end tooling, and integrated.

Why Angular?

- It makes mobile apps easier to handle things and improves performance, load time, etc.
- Angular is component driven. The complexity of the core AngularJS will be removed, resulting better performance.
- Angular targets ES6 and make harder for any hacks or workarounds which ensures the security of the particular business domain.



Angular Features

- **Browser Compatibilty**:
  - It supports IE 9, 10, 11, Firefox, Chrome, Safari, Android 4.1 & Microsoft Edge.
- **Cross platform:**
  - It can run on desktop, mobile, Android, iOS, etc.
- **Development**
  – It provide full support for CS5, TypeScript, Dart, ES6, other languages that compile to JavaScript.
- **Mobile Support**
  – Angular.0 is mobile oriented architecture. There are libraries i.e. Native script which helps mobile development fast.
- **TypeScript**
  – TypeScript(TS) is used heavily in Angular. Google currently using DART for coding. DART or TypeScript can be used for Angular
- **No $Scope in Angular**
  Angular is not using $scope anymore to glue view and controller

Typescript

- Typescript is basically a compiled type language with a strongly typed layer in conjunction with JavaScript.
- Typescript allows to write a class, interface, and module statements just like in Java or C# which boosts the performance of the web and mobile solution as the code written in Typescript are less inclined to run-time errors.

TypeScript code:

```
var message:string =“Hello World”
Console.log(“message”)
```

Typescript saved with the extension .ts

Compiled JavaScript Code:

```
var message =“Hello World”
Console.log(“message”)
```

The file is compiled to Test.js

For more about Typescript

- https://www.learnhowtoprogram.com/javascript/angular-js/typescript-introduction-and-installation

Angular Architecture

- The major 8 blocks of Angular architecture:

Module

Component

Template

Metadata

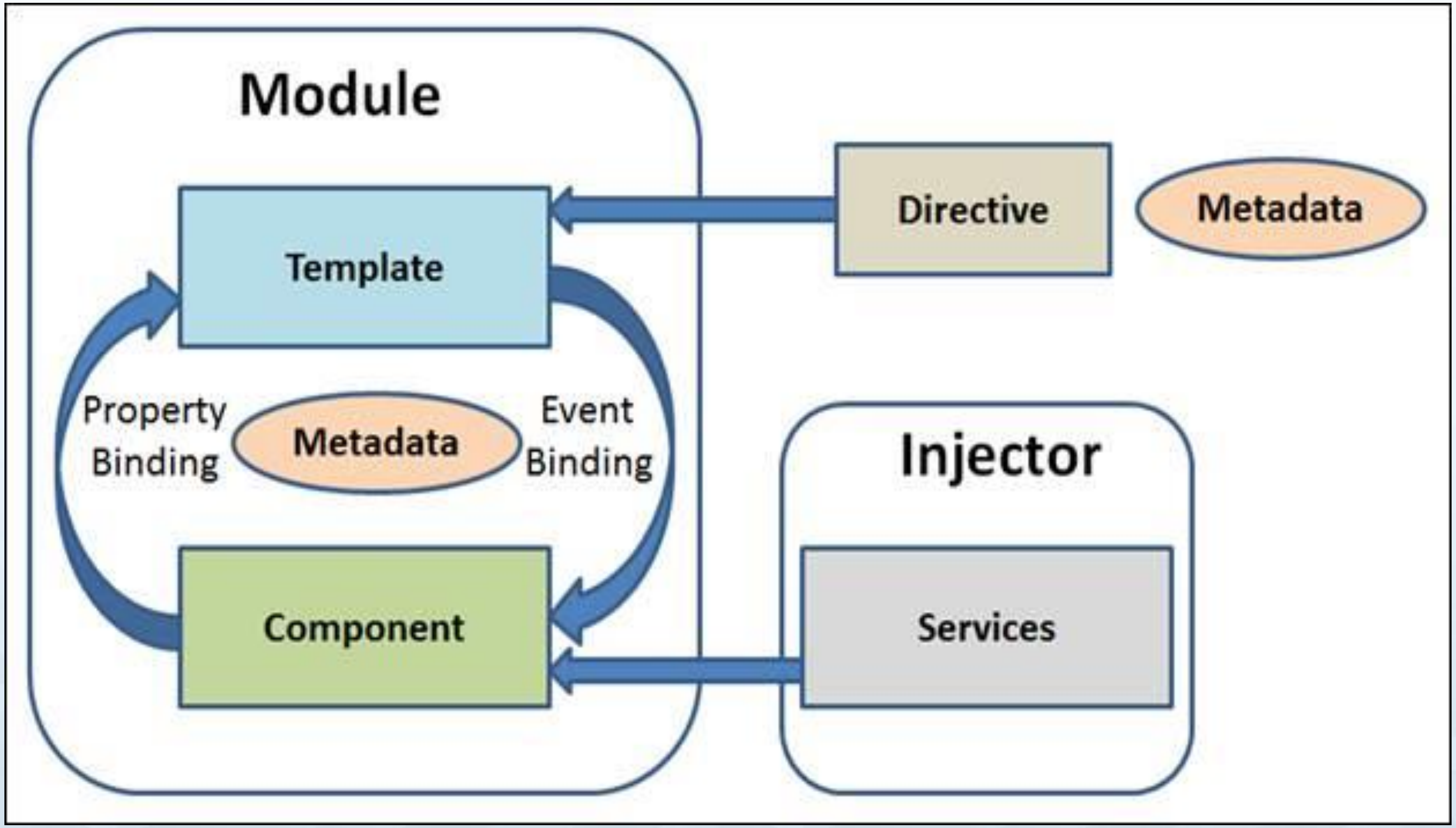
Data Binding

Directive

Services

Dependency Injection

Angular Architecture



NgModules

- The basic building blocks of an Angular application are *NgModules*, which provide a compilation context for *components*.
- NgModules collect related code into functional sets; an Angular app is defined by a set of NgModules.
- An app always has at least a *root module* that enables bootstrapping, and typically has many more *feature modules*.

Component

- Components define *views*, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.
- Every app has at least a root component.
- Components use *services*, which provide specific functionality not directly related to views.
- Service providers can be *injected* into components as *dependencies*, making your code modular, reusable, and efficient.

Component

- Every Angular application has at least one component, the root component that connects a component hierarchy with the page DOM.
- Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.
- The @Component decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

Benefits of Component-based Architecture

As you can see, this component-based architecture makes our applications more organized and maintainable. Plus, we can potentially reuse these components in various parts of an application or in an entirely different application

Modules

- Every Angular app has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application.
- An app typically contains many functional modules.
- Like JavaScript modules, NgModules can import functionality from other NgModules, and allow their own functionality to be exported and used by other NgModules.
- For example, to use the router service in your app, you import the Router NgModule.

First Angular Application Cont…

- Install the latest version of Node.
- Node comes with a tool called **Node Package Manager** or **NPM**
- NPM is used to install Angular CLI.
- Run the following command to install Angular CLI

```
npm install -g @angular/cli
```

The **-g** flag stands for global. If you don't put -g here, Angular CLI will be installed only in the current folder, and it's not going to be accessible anywhere else.

First Angular Application Cont…

- To create a new Angular project use the following command:

  ```
  ng new hello-world
  ```

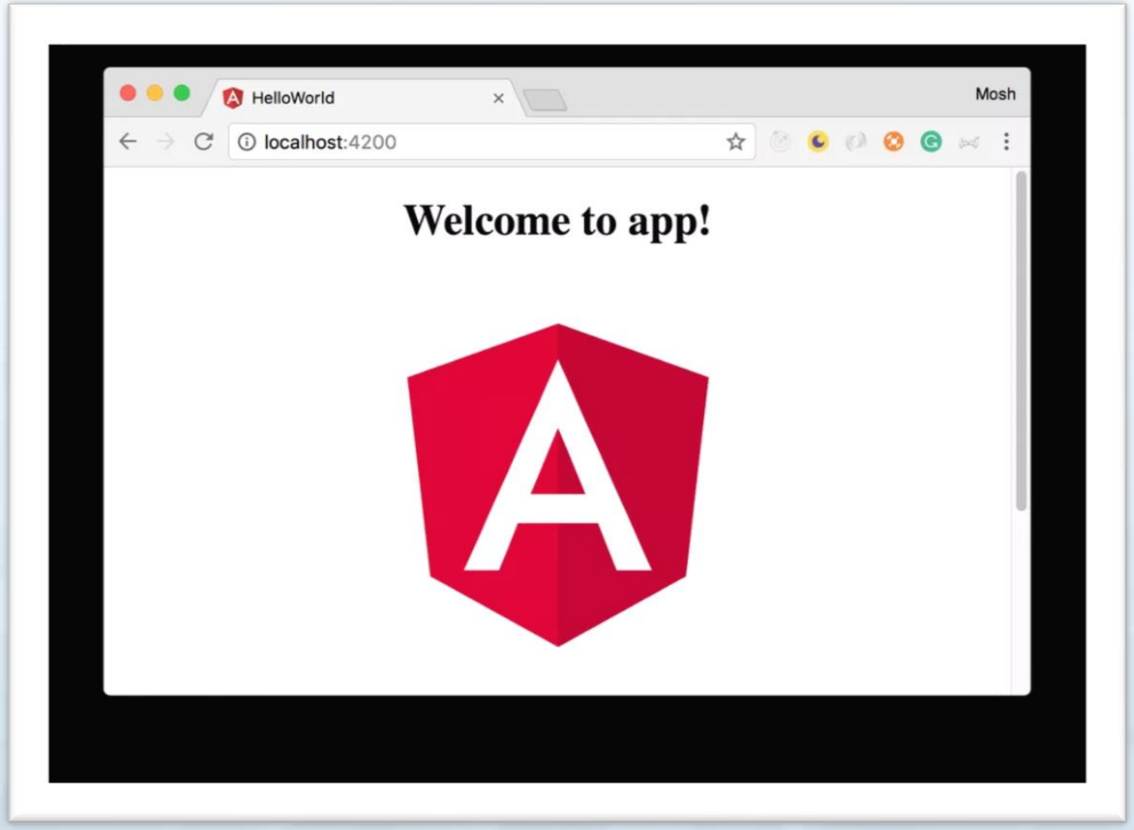
- Angular CLI can be accessed using **ng**.
- Angular CLI will generate a new project called “hello-world” and store it in a folder with the same name

First Angular Application Cont…

- Run the following commands in the terminal:
  - ➢ cd hello-world
  - ➢ npm install
  - ➢ ng serve
- The command **npm install** will install all the dependencies of the application.
- The command **ng serve** compiles the application and hosts it using a lightweight web server.
- Access the application at **http://localhost:4200.**

First Angular Application

Open the browser and navigate to this address **http://localhost:4200.**



Welcome to app!

Structure of Angular Projects Cont…

- Inside the generated folder, the following top-level folders are present:
- **e2e**: includes end-to-end tests.
- **node_modules**: all the third-party libraries that our project is dependent upon.
- **src**: the actual source code of our Angular application. 9.9% of the time you'll be working with the files inside the **src** folder.
- **angular-cli.json:** a configuration file for Angular CLI. This file is used to import third-party stylesheets or define additional environments (eg testing environment) for our application.

Structure of Angular Projects Cont…

- **package.json:** a standard file for Node-based projects. It contains metadata about our project, such as its name, version as well as the list of its dependencies.
- **protractor.conf.js:** Protractor is a tool for running end-to-end tests for Angular projects.
- **karma.conf.js:** Karma is a test runner for JavaScript applications. This file contains some configuration for Karma.

Structure of Angular Projects Cont…

- **tsconfig.json:** includes setting for the TypeScript compiler.
- **tslint.json:** includes the settings for TSLint which is a popular tool for linting TypeScript code.
- It checks the quality of our TypeScript code based on a few configurable parameters.
- This is especially important in a team environment to ensure that everyone follows the same conventions and produces code of the same quality.

Angular Component in Action

Open the **src/app** folder

- app.component.css
- app.component.html
- app.component.spec.ts
- app.component.ts
- app.module.ts

Angular Component in Action

- Each component in an Angular project is physically implemented using four files:
- **A CSS file**: where we define all the styles for that component. These styles will only be scoped to this component and will not leak to the outside.
- **An HTML file**: contains the markup to render in the DOM.

Angular Component in Action

- **A spec file**: includes the unit tests.
- **A TypeScript file**: where we define the state (the data to display) and behavior (logic) of our component.
- An app.module.ts file is also present. This file defines the root module of our application and tells angular how to assemble the app.

Creating a Component Cont…

- **Generating a Component Using Angular CLI**

  ng g c product

  g is short for **generate**, **c** is short for **component** and **product** is the name of our component.

- Inside the **src/app** folder, a new folder **product is generated.** Expand this folder
  - product.component.css
  - product.component.html
  - product.component.spec.ts
  - product.component.ts

Creating a Component

Open **product.component.ts**

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-product',
  templateUrl: './product.component.html',
  styleUrls: ['./product.component.css']
})
export class ProductComponent implements OnInit {
  constructor() { }
  ngOnInit() {
  }
```

Creating a Component

we have a TypeScript *class* called **ProductComponent**.

What is a Class?

- A class is a fundamental building block of many object-oriented programming languages. It’s a container for a bunch of related functions and variables.
- In **product.component.ts**, there is a class called **ProductComponent**.
- This class has 2 functions (methods): **constructor** and **ngOnInit**.

Creating a Component

- **Constructor** is a reserved keyword in TypeScript. A method by that name is a special method in a class.
- This method is called automatically when we create an instance of that class.
- **ngOnInit** is a special method in Angular.
- Angular calls this method when it creates an instance of this component and displays it to the user in the browser

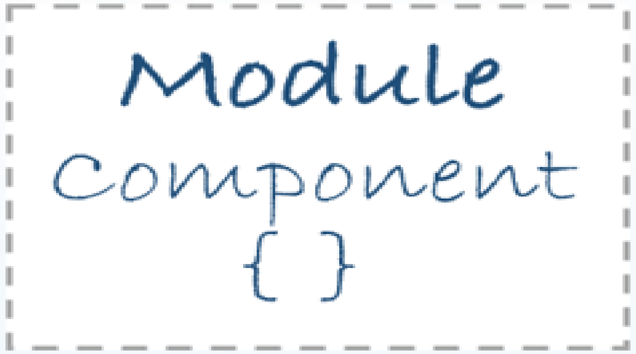
Component Metadata

- we implement a component using a TypeScript class.
- But a class on its own is just a class. It only includes some data and logic for a view. It doesn’t include any HTML markup or CSS styles.
- In order to attach these to this class, we need to promote this class to a component.
- We can do this by using the **@Component()** decorator function on top of this class

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-product',
  templateUrl: './product.component.html',
  styleUrls: ['./product.component.css']
})
export class ProductComponent {
}
```

- It takes an object with the following properties:
- selector
- templateUrl
- styleUrls
- You’ve seen the **selector** in action. It associates a new HTML element to this component.
- The other 2 properties (**templateUrl** and **styleUrls**) are self-explanatory. They specify the path to the HTML template and CSS file(s) for this component.

Module



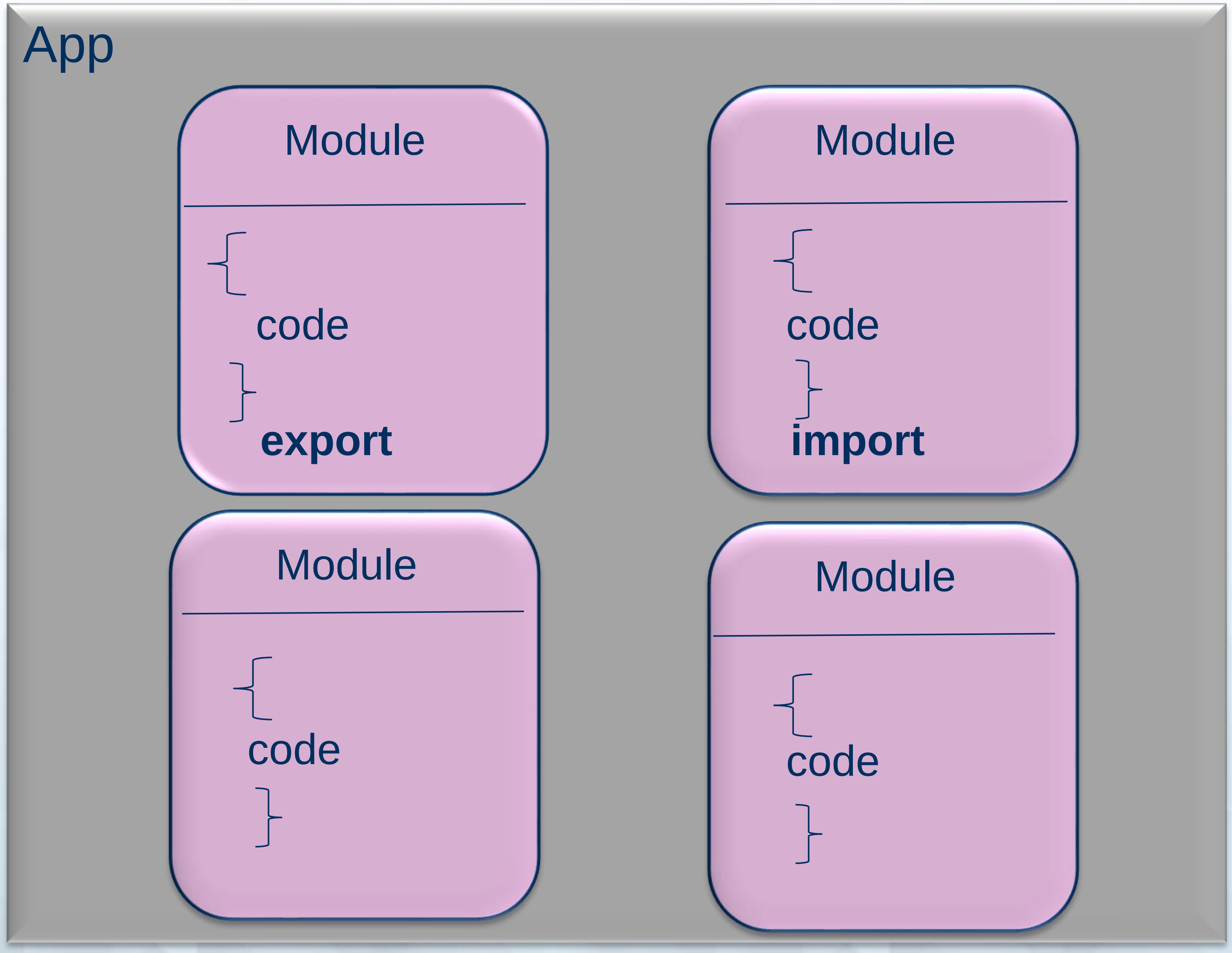
- Module is very similar to a class.
- A module can be described by a block of code which is used to perform a particular single task.
- Angular has a feature of modularity, where a single application is built by separating it in many modules.

- Export statement is used to export component class from a module.

```
export class AppComponent
{
…..
}
```

Module (cont..)

- Modules



Module (cont.)

- NgModule decorator to defines the modules.

```
import  {NgModule} from ‘@angular/core’;

@NgModule({
          imports: [ …],
          declarations: [ … ],
          bootstrap: [ … ]
})

export class AppModule{}
```

Decorator

NgModule

declarations	- the *view classes* that belong to this module. Angular has three kinds of view classes: components, directives, pipes.
exports	the subset of declarations that should be visible and usable in the component templates of other modules.
imports	other modules whose exported classes are needed by component templates declared in *this* module.
providers	- creators of services that this module contributes to the global collection of services; - they become accessible in all parts of the app.
Bootstrap	the main application view, called the *root component*, that hosts all other app views. - Only the *root module* should set this bootstrap property.

Module (cont.)

- Sample app.module.ts file

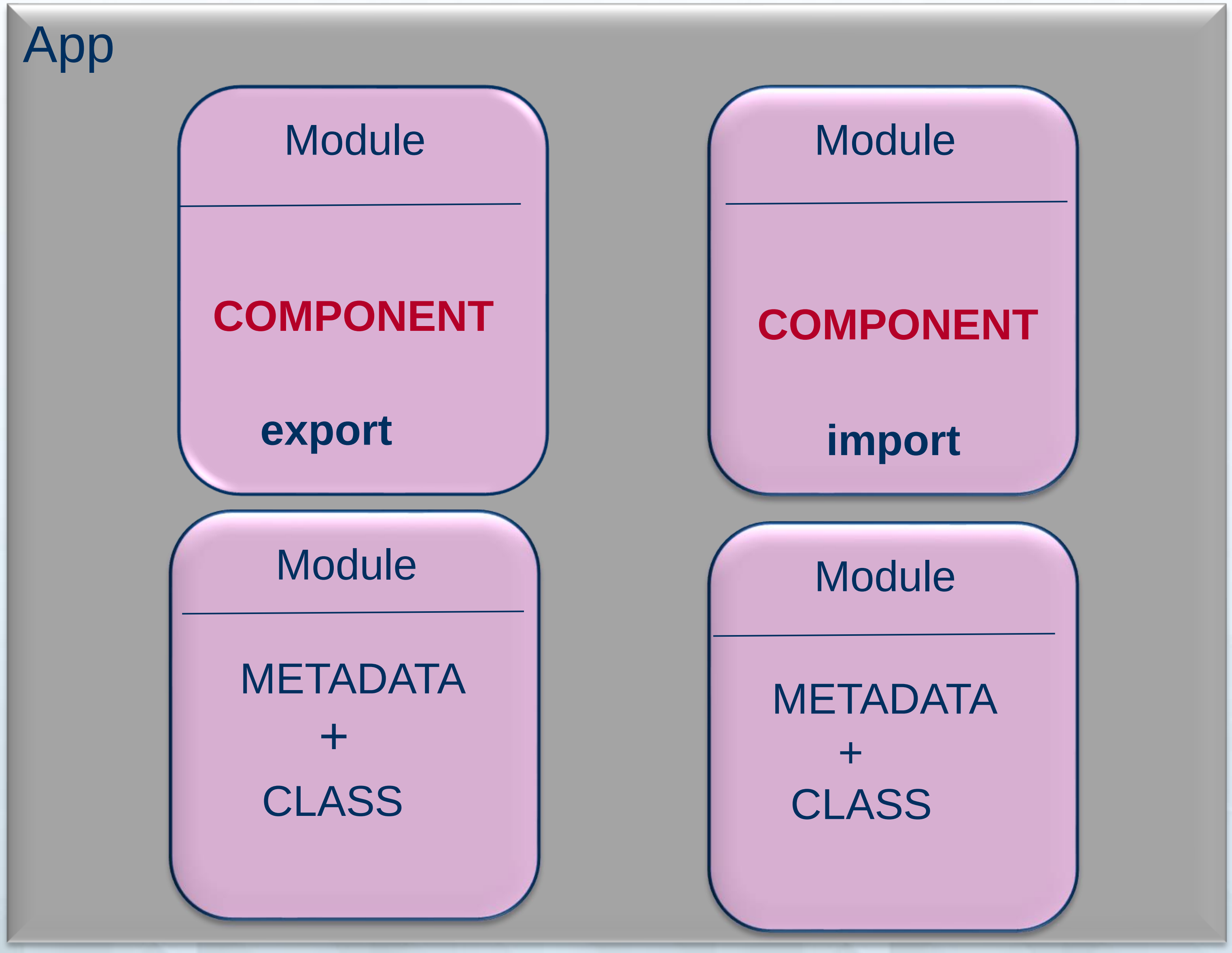
```
import { NgModule }      from    '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { SampleComponent } from "./sample.component";

@NgModule({
  imports:      [ BrowserModule ],
  declarations: [ AppComponent,SampleComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Component

- Components are the most basic building block of an UI in an Angular application.
- @Component is used to register a component and only one component is used per DOM element.
- Component decorator allows you to mark a class as an Angular component and provide additional metadata that determines how the component should be processed, instantiated and used at runtime.

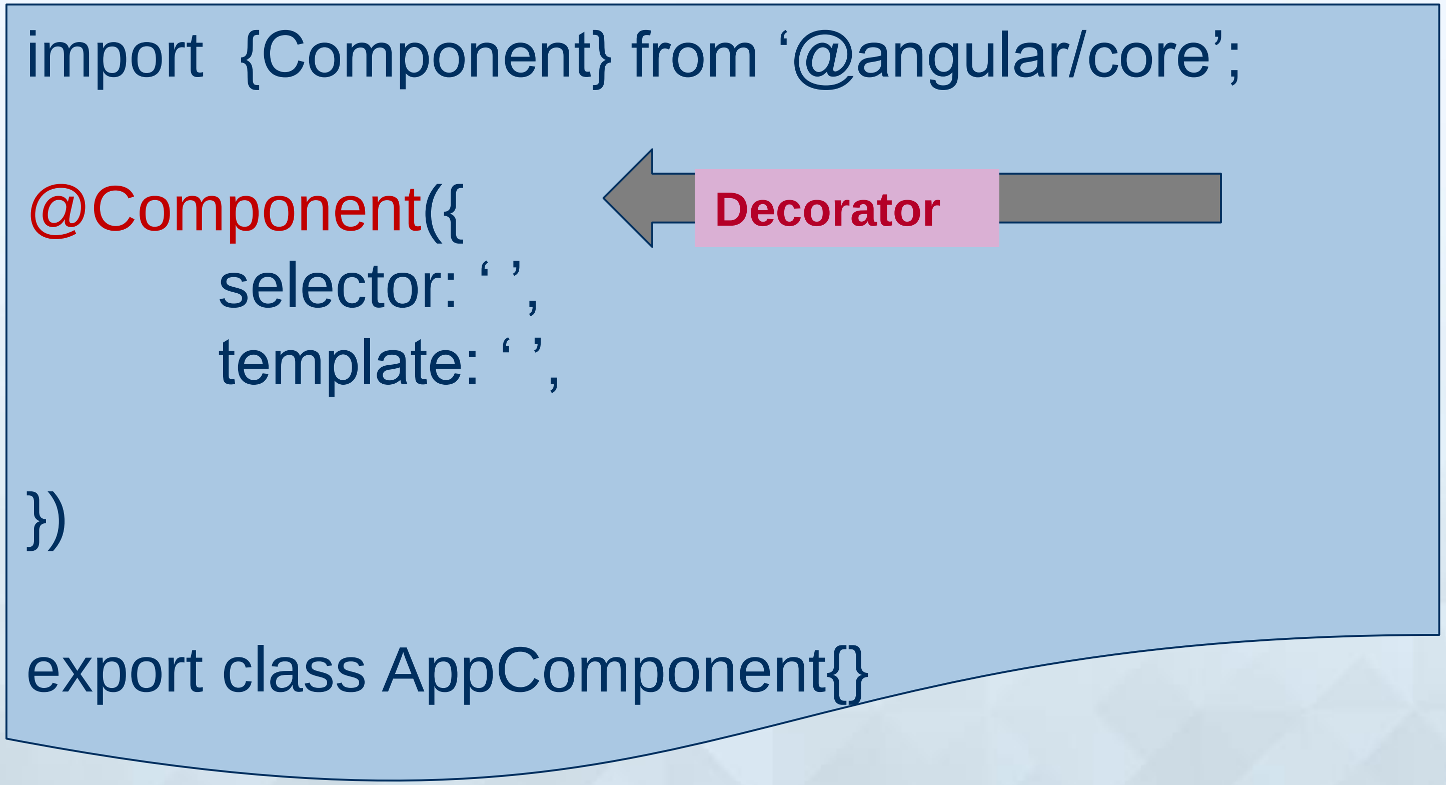
Component (cont..)

- Component



Component (cont.)

- The component would look like.



```
import  {Component} from ‘@angular/core’;

@Component({
          selector: ‘ ’,
          template: ‘ ’,

})

export class AppComponent{}
```

Component (Cont..)

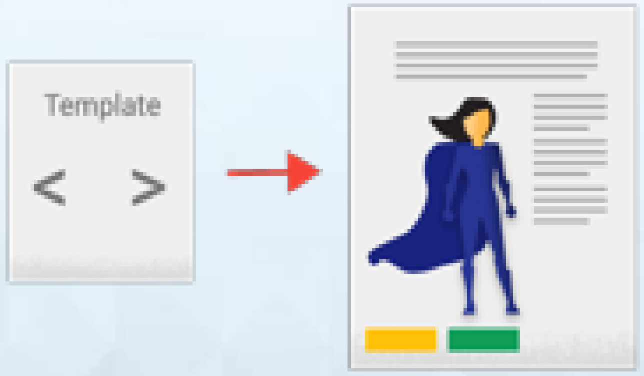
- Sample app.component.ts file

```
import { Component } from '@angular/core';


@Component({
selector: 'my-app',
template: `<h1>Hello {{name}}</h1>
           <sample-app></sample-app> `,
})
export class AppComponent { name = 'Jamuna'; }
```

Template

- Template is the main part which justifies the look of the component.
- It can be said that the view of the component is defined using template.
- To display value, add template expression in code

```
<div>

Your name is : {{name}}

</div>
```



Template (Cont..)

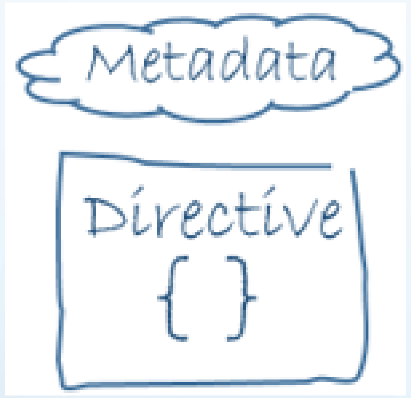
- Template example

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>`,
})
export class AppComponent  { name = 'Jamuna'; }
```

Directive

- Directives are custom HTML attributes used to prolong power of HTML.
- To create a directive, @Directive decorator is applied on connected metadata of the class.

```
import {Directive} from '@angular/core';

@Directive({
selector: 'my-directive',
})
export class MyDirective {
}
```



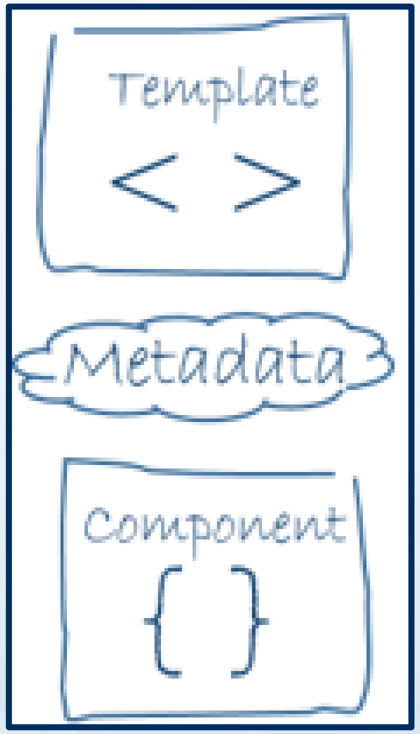
- Directive decorator allows you to mark a class as an Angular directive and provide additional metadata that determines how the directive should be processed, instantiated and used at runtime.
- For More Information about directive
  - https://angular.io/api/core/Directive

Metadata

- Metadata is majorly used to extend the functionality of the class.
- Metadata can be attached to TypeScript using a decorator.

- For example, to define any component in Angular application, use metadata of the class (i.e. @Component decorator)

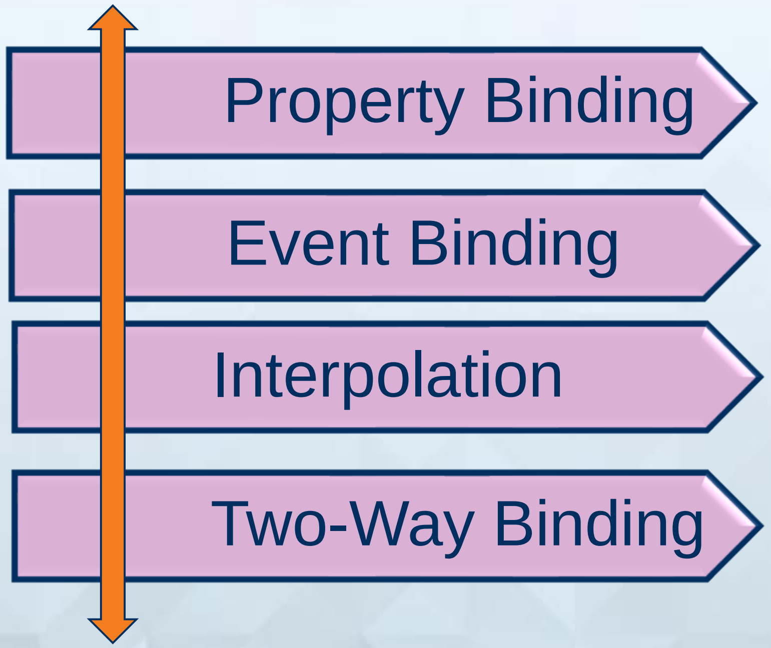
Metadata (cont..)



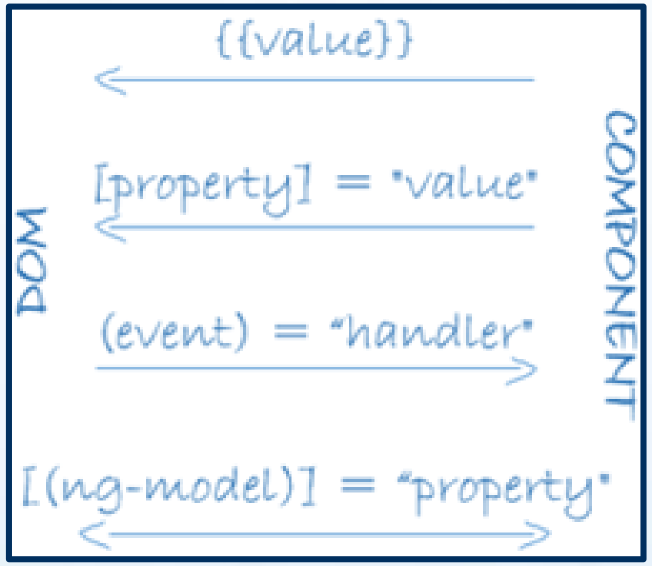
- The metadata in the @Component tells Angular where to get the major building blocks you specify for the component.
- The **template**, **metadata** and the **component** together describes a view.
- @Injectable, @Input and @Output are a few of the more popular decorators.

Data Binding

- The most powerful feature, Data Binding, is the connection bridge between Model and View.
- It gets automatically synchronized.
- Angular.0 supports four types of binding –



Data Binding (Cont..)



The HeroListComponent example:

```
<li>{{hero.name}}</li>

<hero-detail [hero]="selectedHero"></hero-detail>

<li (click)="selectHero(hero)"></li>
```

Interpolation

Property Binding

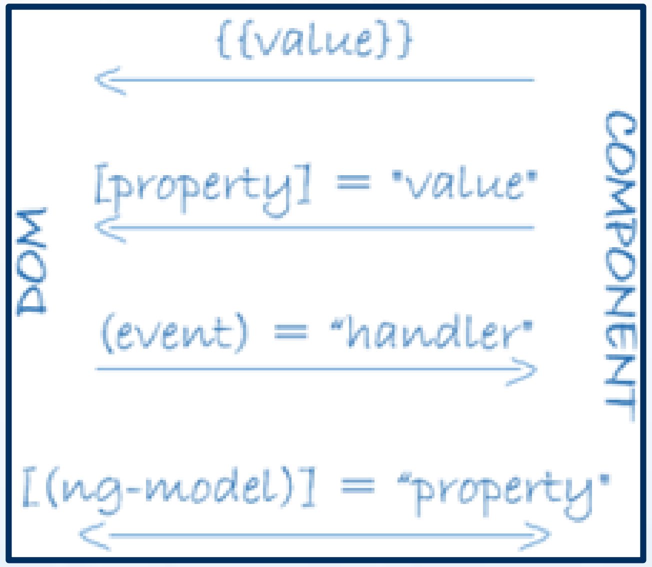
Event Binding

It displays the hero.name property value within the <li>element.

The [hero] property binding the value of selectedHero from the parent HeroListComponent to the hero property of the child HeroDetailComponent.

The (click) event binding calls the component's selectHero method when the user clicks a hero's name.

Data Binding (Cont..)



It combines property and event binding in a single notation, using the **ngModel** directive.

```
The HeroListComponent example:
<input [(ngModel)]="hero.name">
```

In two-way binding, a data property value flows to the input box from the component as with property binding. The user's changes also flow back to the component, resetting the property to the latest value, as with event binding.

Property binding & Event binding

- **<p>{{ title }}</p>**
- **<p>{{ itemCount }}</p>**

- **Event Binding**
- Similar to property binding, we have another concept in Angular called *event binding*. With property binding, we bind properties of DOM elements to fields/properties in our component. With event binding, we bind events of DOM elements (such as clicks) to methods in our component. So, for example, when the user clicks on a button, a method in our component will be called.
- <p>{{ title }}</p>
- <p>{{ itemCount }}</p>
- <button (click)="addItem()">Add</button>

Data Binding (Cont..)

- Data Binding Demos

Interpolation and property binding

Event Binding

TwoWayBinding

Getting Started

```
my-app
 ├→ e2e
 │   ├→ app.e2e-spec.ts
 │   ├→ app.po.ts
 │   └→ tsconfig.e2e.json
 ├→ node_modules/...
 ├→ src/...
 ├→ .angular-cli.json
 ├→ .editorconfig
 ├→ .gitignore
 ├→ karma.conf.js
 ├→ package.json
 ├→ protractor.conf.js
 ├→ README.md
 ├→ tsconfig.json
 └→ tslint.json
```

- For information about quickstart directives:

  https://angular.io/guide/quickstart

To set up development environment:

```
npm install
```

To start the application:

```
npm run serve
npm start
```

Quick Start Template

quickstart template

Demos

HelloWorldDemo

Creating ComponentDemo

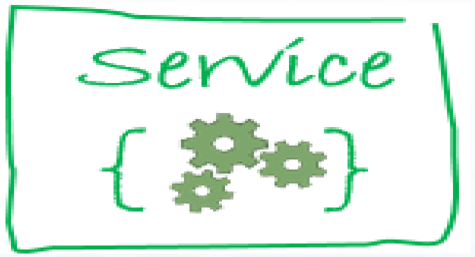
Adding Styles to Components

TemplateURL Demo

Array&Loopdemo

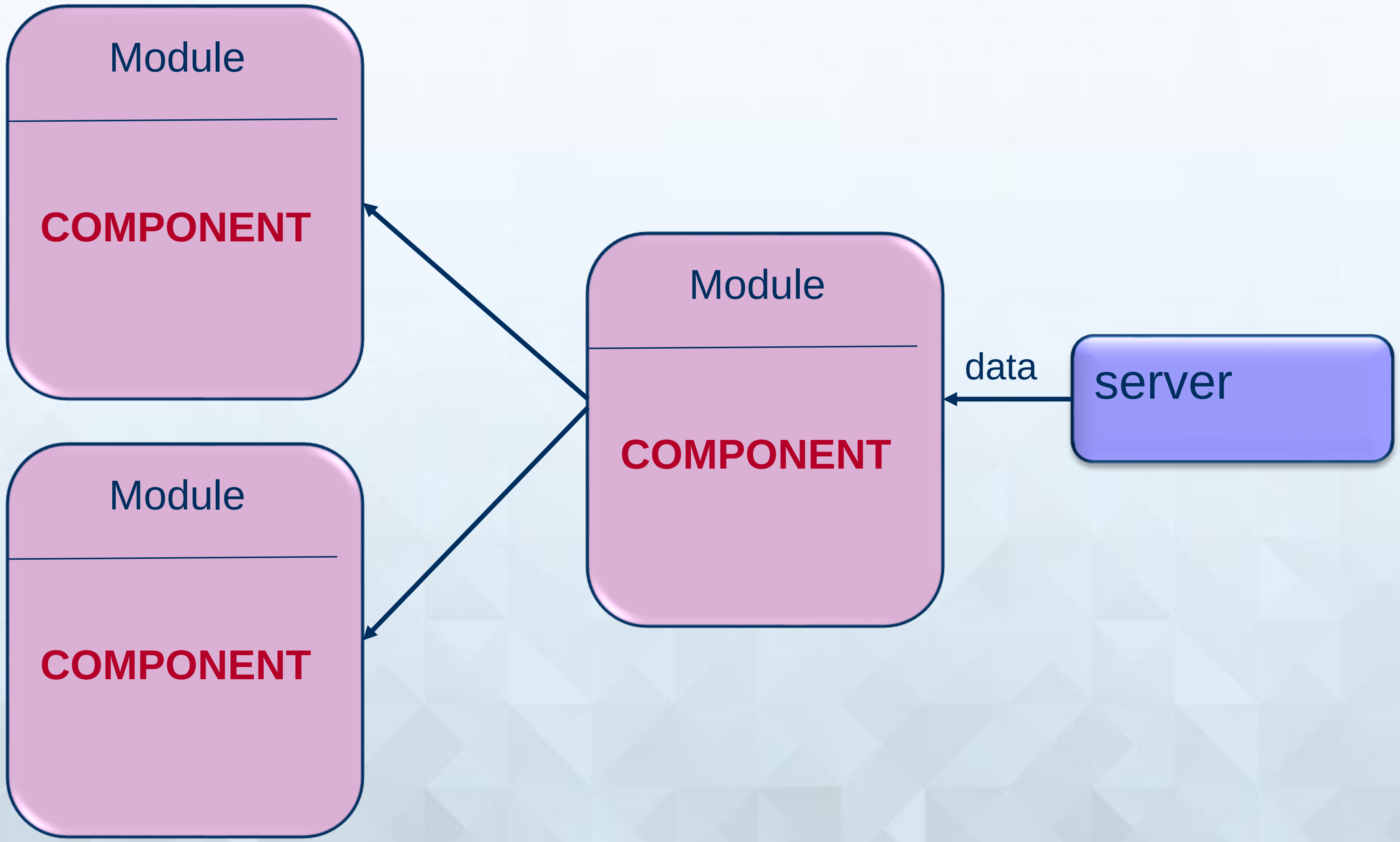
Pipe Demo

Services

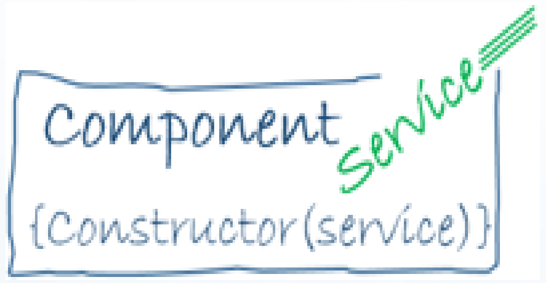


- Services are used when a single functionality is used commonly in various modules of the application.
- Basically, is used to share the data and behavior within the application.
- Service has no base class.
- Commonly used services are **logging service**, **data service**, **message service**, etc.

Services (Cont..)

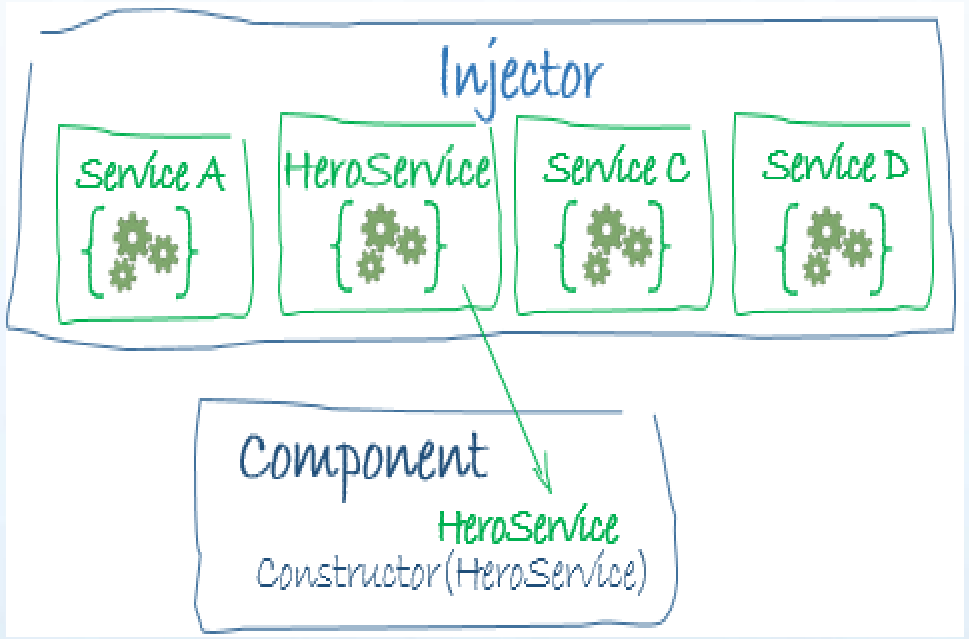


Dependency Injection



- To attach the functionality of components at runtime, dependency injection is used.
- As objects are passed as dependencies, it makes dependencies configurable - removing its hard-coded dependencies.
- With dependency injection, components can be easily maintainable, reusable and testable.

Dependency Injection (Cont..)



```
constructor(private service:HelloService){}
```

- When Angular creates a component, it first asks an injector for the services that the component requires.
- An injector maintains a container of service instances that it has previously created. If a requested service instance is not in the container, the injector makes one and adds it to the container before returning the service to Angular.
- When all requested services have been resolved and returned, Angular can call the component's constructor with those services as arguments. This is *dependency injection*.

Dependency Injection (Cont..)

Points to remember about dependency injection:

- Dependency injection is wired into the Angular framework and used everywhere.
- The *injector* is the main mechanism.
  - An injector maintains a ***container*** of service instances that it created.
  - An injector can create a new service instance from a ***provider***.
- A ***provider*** is a recipe for creating a service.
- Register *providers* with **injectors**.

Routers

- The Angular Router enables navigation from one view to the next as users perform application tasks
- Enter a URL in the address bar and the browser navigates to a corresponding page.
- Click links on the page and the browser navigates to a new page.
- Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.

```
import  { RouterModule, Routes } from  '@angular/router';
```

References

For more information about architecture:

- https://angular.io/guide/architecture

For Typescript

- https://www.learnhowtoprogram.com/javascript/angular-js/typescript-introduction-and-installation

HEXAWARE

Innovative Services

Passionate Employees

Delighted Customers

Thank you

www.hexaware.com