



Angular Part – 1

PROF. P. M. JADAV
ASSOCIATE PROFESSOR
COMPUTER ENGINEERING DEPARTMENT
FACULTY OF TECHNOLOGY
DHARMSINH DESAI UNIVERSITY, NADIAD

Content

- Angular Introduction
- Architecture
- Modules
- Services
- Dependency Injection
- Angular Project Files

Content

- Component Creation
- Component Template
- Component Selectors (Element, Class, Attribute)
- Interpolation
- Property Binding
- HTML Attribute Vs. DOM Property

Content

- Class Binding
- Style Binding
- Event Binding
- Template Reference Variable

Angular Introduction

- Framework to build client side applications
- Used to develop SPAs (Single Page Applications)
- Modular approach
- Re-usable code (components)
- Quick and easier development
- Unit testable and easily maintainable

Angular History

- 2010 – AngularJS
- 2016 – Angular 2
- 2016 Dec – Angular 4
- 2017 Nov – Angular 5
- 2018 May – Angular 6

Semantic Versioning

Angular 6.0.3

Major

Minor
(Don't break any
functionality)

Fix (Patch)

```
graph BT; Major --> Angular[Angular 6.0.3]; Minor --> Angular; Fix[Fix (Patch)] --> Angular;
```

Development Environment

- Node
- NPM (installed as part of Node)
- Angular CLI
- IDE (Visual Studio Code/Atom/Sublime)

Checking Versions

- `node -v`
- `npm -v`
- `ng v`

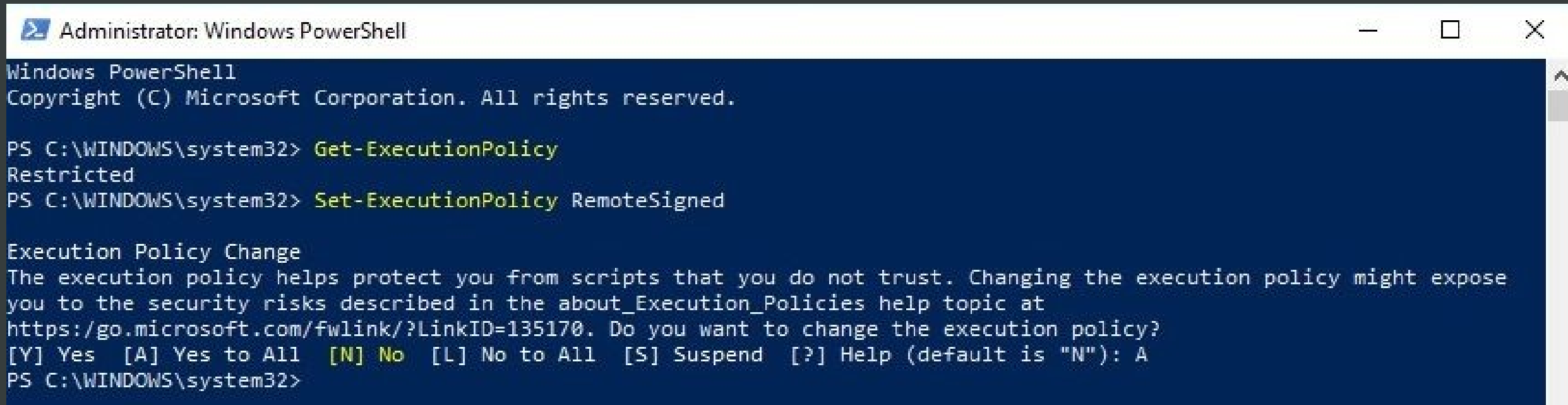
Angular CLI (Create New Angular App)

- `npm install -g @angular/cli`
- `ng new my-app`
- ✓ Enter "**N**" for **routing**
- ✓ Select "**CSS**" as the default **styles**

Angular CLI (Run the Angular App)

- `cd my-app`
- `ng serve --open`
- (If you encounter an error due to Windows security policies see the next slide)

Angular CLI (Windows Security Issues)



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\WINDOWS\system32> Get-ExecutionPolicy
Restricted
PS C:\WINDOWS\system32> Set-ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
https://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): A
PS C:\WINDOWS\system32>
```

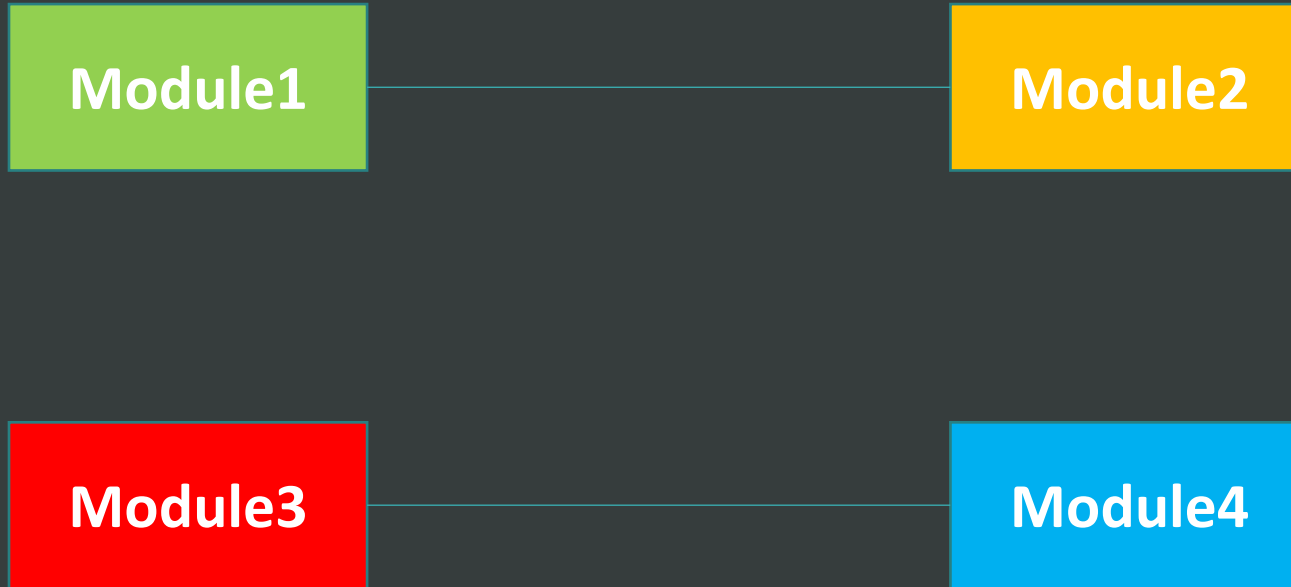
Architecture

- Modules
 - User, Admin
 - Root Module – App Module

Architecture

- Components
 - Part of modules
 - HTML & CSS (view) + Class (logic)
 - Components for different views
 - Navbar, sidebar, main content
 - Root component – App component

Modules



Modules



The diagram illustrates a module structure. It features a large yellow rounded rectangle in the center. Inside this rectangle, there are two smaller teal rounded rectangles stacked vertically. The top teal rectangle contains the word 'Component' in white text, and the bottom teal rectangle contains the word 'Services' in white text. The entire diagram is set against a dark gray background.

Component

Services

Services

- **Service** is a broad category encompassing any value, function, or feature

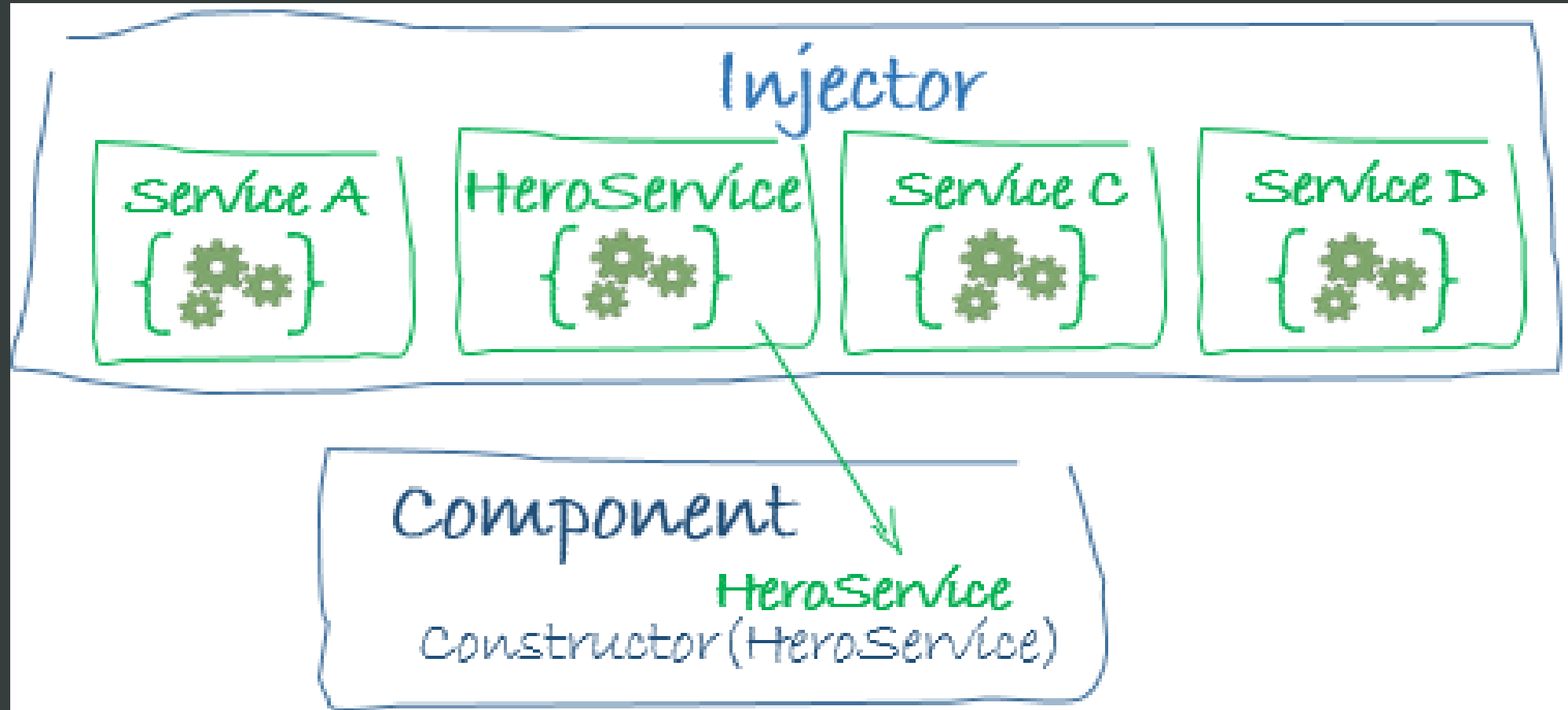
that your application needs

- **Examples** include:
 - logging service
 - data service
 - message bus
 - tax calculator
 - application configuration

Dependency Injection

- *Dependency injection* is a way to supply a new instance of a class with the fully-formed dependencies it requires
- Most dependencies are **services**
- Angular uses dependency injection to provide new components with the services they need.

Dependency Injection



Client Code Without Dependency Injection

```
public class Client {  
    private ExampleService service;  
  
    Client() {  
        service = new ExampleService();  
    }  
  
    public String greet() {  
        return service.getName();  
    }  
}
```

Interface Declaration

```
interface IService
{
    String doServe();
}
class ExampleService implements IService
{
    public String doServe() {
        return "What can I do for you?";
    }
}
```

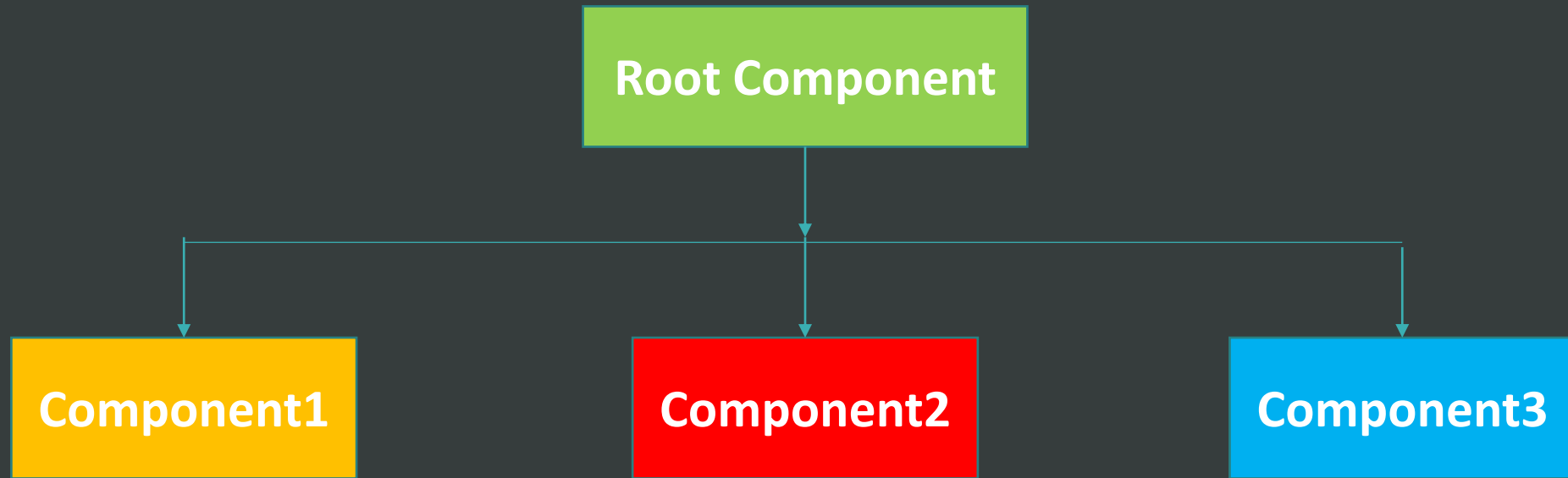
Client Code With Dependency Injection

```
class Client {  
    private IService serv;  
  
    Client(IService serv) {  
        this.serv = serv;  
    }  
  
    public String greet() {  
        return serv.doServe();  
    }  
}
```

Injector Code

```
public class Injector {  
    public static void main(String[] args) {  
        IService service = new ExampleService();  
  
        Client client = new Client(service);  
  
        System.out.println(client.greet());  
    }  
}
```

Components



Architecture Summary

- Angular app – one or more modules
- Modules – one or more components and services
- Components – HTML + Class
- Services – Business logic
- Modules render the view in the browser

Angular Project Files

| Filename | Description |
|--------------------|---|
| main.ts | Entry point to the Angular app (renders AppModule which is root module) |
| package.json | All the metadata for the project and all the dependencies (project dependency + developer dependency) |
| node_modules | Installed modules in the project |
| app.module.ts | Root module of the application |
| app.component.ts | Root component of the application |
| app.component.html | View for app component |

Angular Project Files

| Filename | Description |
|-------------------|---|
| app.component.css | Styles for app component |
| tsconfig.app.json | TypeScript configuration file |
| index.html | Home page |
| pollyfills.ts | Scripts to support different browsers |
| tslint.json | Configuration for tslint (linting i.e. grammar checking in editor) |
| | |
| | |

NgModule

- Modules are a great way to **organize** an application and extend it with capabilities from external libraries
- **Angular libraries** are **NgModules**, such as **FormsModule**, **HttpClientModule**, and **RouterModule**
- Many **third-party libraries** are available as NgModules such as **Material Design**, **Ionic**, and **AngularFire2**

NgModule

- NgModules **consolidate components, directives, and pipes** into cohesive blocks of functionality, each focused on a feature area, application business domain, workflow, or common collection of utilities
- Modules can **add services** to the application (Such services might be internally developed or come from outside sources, such as the Angular router and HTTP client)
- Modules can be **loaded eagerly** when the application starts or **lazy loaded** asynchronously by the router

NgModule Metadata

```
@NgModule({  
  declarations: [  
    AppComponent, TestComponent, MypowerPipe  
  ],  
  imports: [  
    BrowserModule, FormsModule  
  ],  
  providers: [],  
  bootstrap: [AppComponent]  
})
```

NgModule Metadata

- **Declares** which components, directives, and pipes belong to the module
- Makes some of those components, directives, and pipes public (**exports modules**) so that other module's component templates can use them
- **Imports** other **modules** with the components, directives, and pipes that components in the current module need
- **Provides services** that the other application components can use

NgModule Class Vs. JavaScript Modules

1. An NgModule **bounds declarable classes** only. Declarables are the only classes that matter to the Angular compiler.
2. Instead of **defining all member classes** in one giant file as in a JavaScript module, you list the module's classes in the `@NgModule.declarations` list.

NgModule Class Vs. JavaScript Modules

3. An NgModule can **only export the declarable classes it owns or imports** from other modules. It doesn't declare or export any other kind of class.
4. Unlike JavaScript modules, an NgModule can **extend the entire application with services by adding providers** to the `@NgModule.providers` list.

Component

Template

View
in
HTML

Class

Code
in
TypeScript

Metadata

Information
as
Decorator

main.ts

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-
dynamic';
import { AppModule }      from './app/app.module';
import { environment }     from './environments/environment';

if (environment.production) {
    enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule).catch(err =>
console.log(err));
```

index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>FirstApp</title>
  <base href="/"> <meta name="viewport" content="width=device-
width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

app.module.ts

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

```
@NgModule({  
  declarations: [ AppComponent ],  
  imports:      [ BrowserModule ],  
  providers:    [],  
  bootstrap:    [AppComponent]  
})
```

```
export class AppModule { }
```

app.component.ts

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

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

```
export class AppComponent {  
  title = 'My first Angular application';  
}
```

app.component.html

```
<div style="text-align:center">  
  <h1>  
    Welcome to {{ title }}.  
  </h1>  
</div>
```

Build and Deploy the App (ng serve)



Creating and Using Angular Services

- ng g s student

g – generate

s – service

student – service name

student.service.ts

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

@Injectable({
  providedIn: 'root'
})
export class StudentService {
  public students_list : string[] = ['Rajesh', 'Mahesh',
  'Pritesh', 'Kalpesh']
  constructor() { }
  getStudentsList() : string[] {
    return this.students_list;
  }
}
```

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { StudentService } from '../student.service';
@Component({
  selector: 'ddu',
  templateUrl: '../app.component.html',
  styleUrls: ['../app.component.css']
})
export class AppComponent implements OnInit {
  public students_list: string[]
  constructor(private studentService: StudentService) {
  }
  ngOnInit() {
    this.students_list = this.studentService.getStudentsList()
  }
}
```

app.component.html

```
<ul>
  <li *ngFor="let students of students_list">
    {{ students }}
  </li>
</ul>
```

- Rajesh
- Mahesh
- Pritesh
- Kalpesh

Component Creation

- `ng g c test`

`g` – generate

`c` – component

`test` – component name

Note: This will add an entry of `test` component to root module `"app.module.ts"`. If it is not done, do it manually.

Component Decorator (Template)

test.component.ts

```
@Component({  
  selector: 'app-test',  
  template: "<div> This is an inline template </div>",  
  styleUrls: ['./test.component.css']  
})
```

Component Template

test.component.ts

```
@Component ({  
  selector: 'app-test',  
  template: `<div>  
    This is a test component  
  </div>`,  
  styleUrls: ['./test.component.css']  
})
```



Backtick
character

Component Selector (element)

test.component.ts

```
@Component ({  
  selector: 'app-test',  
  template: `<div>  
    This is a test component  
  </div>`,  
  styleUrls: ['./test.component.css']  
})
```


app.component.html

```
<div style="text-align:center">
```

```
<h1>
```

```
    Welcome to {{ title }}!
```

```
</h1>
```

```
<app-test></app-test>
```

```
</div>
```

Component Selector (class)

test.component.ts

```
@Component ({  
  selector: '.app-test',  
  template: `

This is a test component  
  </div>`,  
  styleUrls: ['./test.component.css']  
})


```

app.component.html

```
<div style="text-align:center">  
  <h1>    Welcome to {{ title }}!</h1>  
  <div class="app-test">  
    </div>  
</div>
```

Component Selector (attribute)

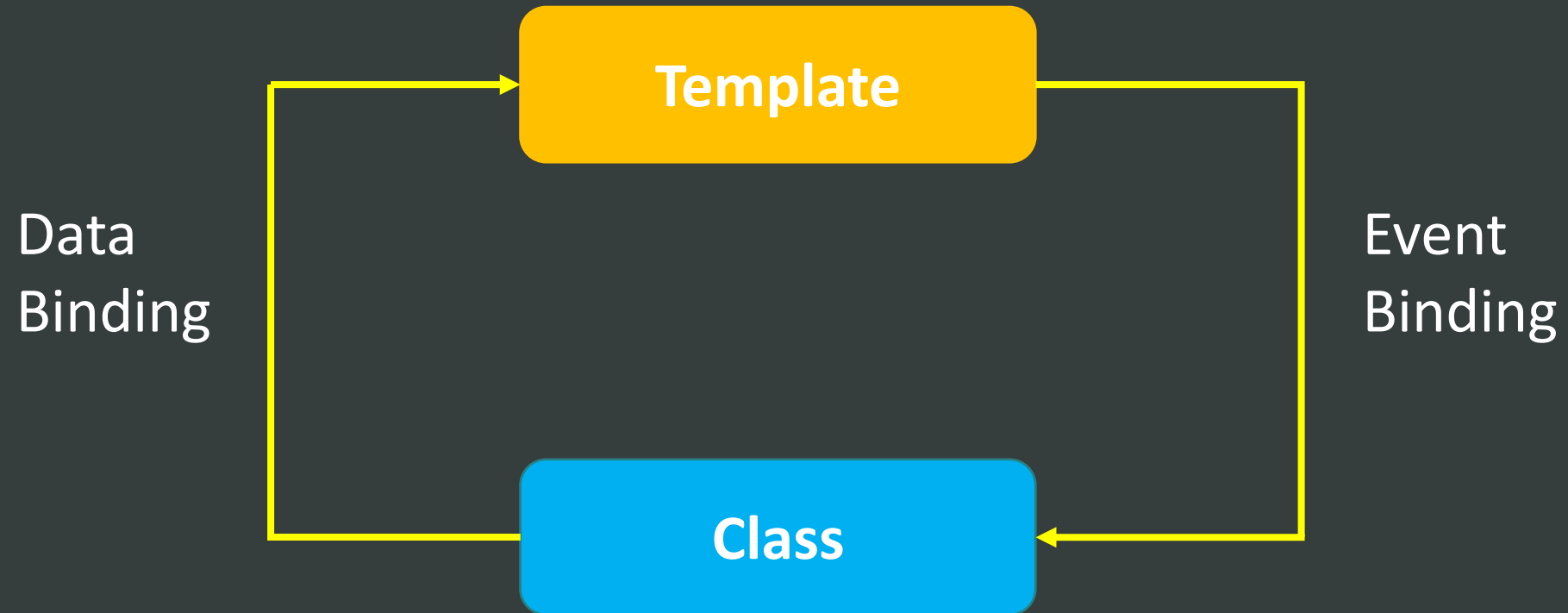
test.component.ts

```
@Component ({  
  selector: '[app-test]',  
  template: `<div>  
    This is a test component  
  </div>`,  
  styleUrls: ['./test.component.css']  
})
```

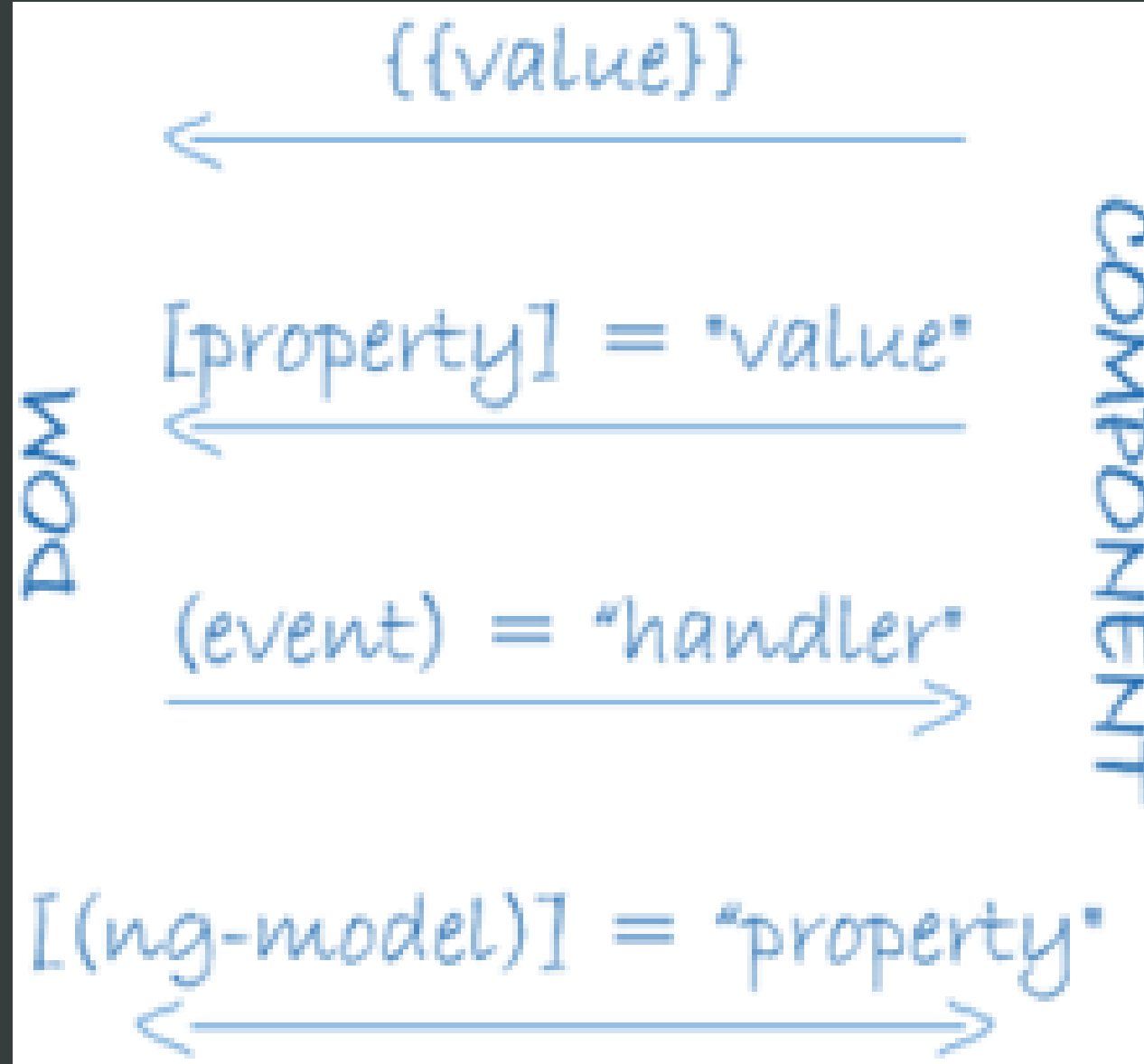
app.component.html

```
<div style="text-align:center">  
  <h1>    Welcome to {{ title }}!</h1>  
  <div app-test>  
  </div>  
</div>
```

Binding



Binding



Interpolation ({{ ... }})

```
export class TestComponent implements OnInit {  
    public name: string = "PMJ"  
    constructor() { }  
    ngOnInit() {}  
    public sayHello():string {  
        return "Hello"  
    }  
}
```


Interpolation

test.component.ts

```
@Component ({  
  selector: 'app-test',  
  template: `

## {{name.length}}{{ sayHello() }} {{ name }} {{ 1+1 }} {{ "a = 2 + 2 assignment will not work in interpolation" }} {{ "window.location.href will not work. Can be used in class method" }} </h2>`, styleUrls: ['./test.component.css'] })


```

app.component.html

```
<div style="text-align:center">  
  <h1>    Welcome to {{ title }}!</h1>  
  <app-test></app-test>  
</div>
```

HTML Attribute Vs. DOM Property

- **Attributes** are defined by **HTML**
- **Properties** are defined by the **DOM** (Document Object Model)
- A few HTML attributes have **1:1 mapping** to properties. **id** is one example.
- Some HTML attributes don't have corresponding properties. **colspan** is one example.
- Some DOM properties don't have corresponding attributes. **textContent** is one example.

HTML Attribute Vs. DOM Property

- Attributes initializes the DOM properties
- **Attribute** values **cannot change** once they are initialized
- **Property** values however **can change**

HTML Attribute Vs. DOM Property

Welcome to Angular

```
>> $('input').getAttribute('value')
```

```
← "DDU"
```

```
>> $('input').value
```

```
← "DDU"
```

Welcome to Angular

```
>> $('input').getAttribute('value')
```

```
← "DDU"
```

```
>> $('input').value
```

```
← "DDIT"
```

HTML Attribute Vs. DOM Property

If JQuery CDN is referenced in index.html

Welcome to Angular

```
>> $('input').attr('value')
```

```
← "DDU"
```

```
>> $('input').val()
```

```
← "DDU"
```

Welcome to Angular

```
>> $('input').attr('value')
```

```
← "DDU"
```

```
>> $('input').val()
```

```
← "DDIT"
```

Property Binding (test.component.ts)

```
@Component ({  
  selector: 'app-test',  
  template: `<div [id]="myId" >Some text </div>  
             <div id={{myId}} >Some text </div>`,  
  styles: [`  
    #testId { font-size: 20px;   color: red;   padding: 20px; }  
  `]  
})
```

Property Binding

test.component.ts

```
export class TestComponent implements OnInit {  
    public myId: string = "testId"  
    constructor() { }  
    ngOnInit() { }  
}
```


Property Binding

- Property binding works with string only
 - It will not work if the property is of other type, say Boolean
- e.g.

```
<input disabled type="text" value="PMJ">
```

- We cannot write disabled="false"
- So using property binding method, we write:

```
<input [disabled]="isDisabled" type="text" value="PMJ">
```

```
<input bind-disabled="isDisabled" type="text" value="PMJ">
```

Class Binding

```
template: ` <h2 class="success">DDU</h2>
          <h2 [class]="resClass">DDU</h2>
          <h2 [class.failure]="hasError">DDU</h2>
          <h2 [ngClass]="textClasses">DDU</h2>      `
styles: [`    .success { color: green;    }
            .failure { color : red;      }
            .special { font-style : italic;    }      `]
```

Class Binding

```
export class TestComponent implements OnInit {  
    public resClass:string = "success"  
    public hasError:boolean = true;  
    public isSpecial : boolean = true;  
    public textClasses = {  
        success : !this.hasError,  
        failure : this.hasError,  
        special : this.isSpecial  
    }  
    ...  
}
```

Style Binding

template: `

```
<h2 [style.color]="blue">Style Binding 1</h2>
```

```
<h2 [style.color]="hasError ? 'red' : 'blue'">Style Binding 2</h2>
```

```
<h2 [style.color]="highlightColor">Style Binding 3</h2>
```

```
<h2 [ngStyle]="titleStyles">Style Binding 4</h2>
```

`

Style Binding

```
export class TestComponent implements OnInit {  
    public hasError:boolean = true  
    public highlightColor : string = "yellow"  
    public titleStyles = {  
        color : "gray",  
        fontStyle : "italic"  
    }  
    ...  
}
```

Event Binding (event)

template:

```
"<button (click) = "onSave()">Save</button>"
```

OR

```
"<button on-click = "onSave()">Save</button>"
```

Canonical
Form



OR

```
"<button on-click = "onSave($event)">Save</button>"
```

Event Binding (event)

template:

```
"<button (click) = "name = 'pmj'">Save</button>"
```

Event Binding (event)

```
export class TestComponent implements OnInit {  
  ...  
  public onSave(event):void {  
    console.log(event)  
    console.log(event.type)  
    console.log(event.target)  
  }  
  ...  
}
```


Event Binding (Ex.)

template:

```
"<input type='text' (keyup)='onKeyUp($event)' >"
```

Event Binding (Ex.)

```
export class AppComponent implements OnInit {  
    onKeyUp(event) {  
        if (event.keyCode == 13) {  
            console.log(event.type)  
            console.log(event.target)  
            console.log("ENTER key pressed.")  
        }  
    }  
}
```

Event Filtering (Ex.)

template:

```
`<input type="text" (keyup.enter)="onKeyUp()">`
```

Event Filtering (Ex.)

```
export class AppComponent implements OnInit {  
    onKeyUp(event) {  
        if (event.keyCode == 13) {  
            console.log(event.type)  
            console.log(event.target)  
        }  
    }  
}
```

Template Reference Variable

template: `

```
<input #myId type="text">
```

```
<button (click)="logName(myId)">
```

Log Element

```
</button>
```

```
<button (click)="logName(myId.value)">
```

Log Value

```
</button>
```

`

Template Reference Variable

```
export class TestComponent implements OnInit {  
    ...  
    public logName(value):void {  
        console.log(value)  
    }  
    ...  
}
```

References

- <https://angular.io/docs>