Unit-4

Introduction to Angular

- 4.6 Introduction to Angular Concepts
- 4.7 Angular Architecture
- 4.8 Angular Components
- 4.9 Component Life Cycle
- 4.10 Data Binding in Angular
- 4.11 Component Communication
- 4.12 Angular Directives
- 4.13 Angular Routing

4.6 INTRODUCTION

- **Angular** ("**Angular 2" or "Angular CLI**") is a TypeScript-based free and open-source web application framework led by the Angular Team at Google.
- Angular is used as the **frontend** of the MEAN stack and FULL stack.
- Angular is **written in TypeScript**.
- Angular is a platform and framework for building **single-page client applications (SPAs)** using HTML and TypeScript.
- Angular internally follows **MVC** (Model-View-Controller) architecture.
- **Evolved from AngularJS** and is a complete rewrite of the framework.

- designed from the beginning with **mobile support** so you can easily address mobile platforms, and **also provides server-side rendering** of web application on the browsers – "**Progressive Web Applications**"
  - app that's **built using web platform technologies**, but that provides a **user experience like that of a platform-specific app**.
- Latest version of Angular – **Version 17**
- Angular is a development platform, built on TypeScript. As a platform, Angular includes:
  - **A component-based framework** for building scalable web applications.
  - **A collection of well-integrated libraries** that cover a wide variety of features, including routing, forms management, client-server communication, and more
  - **A suite of developer tools** to help you develop, build, test, and update your code

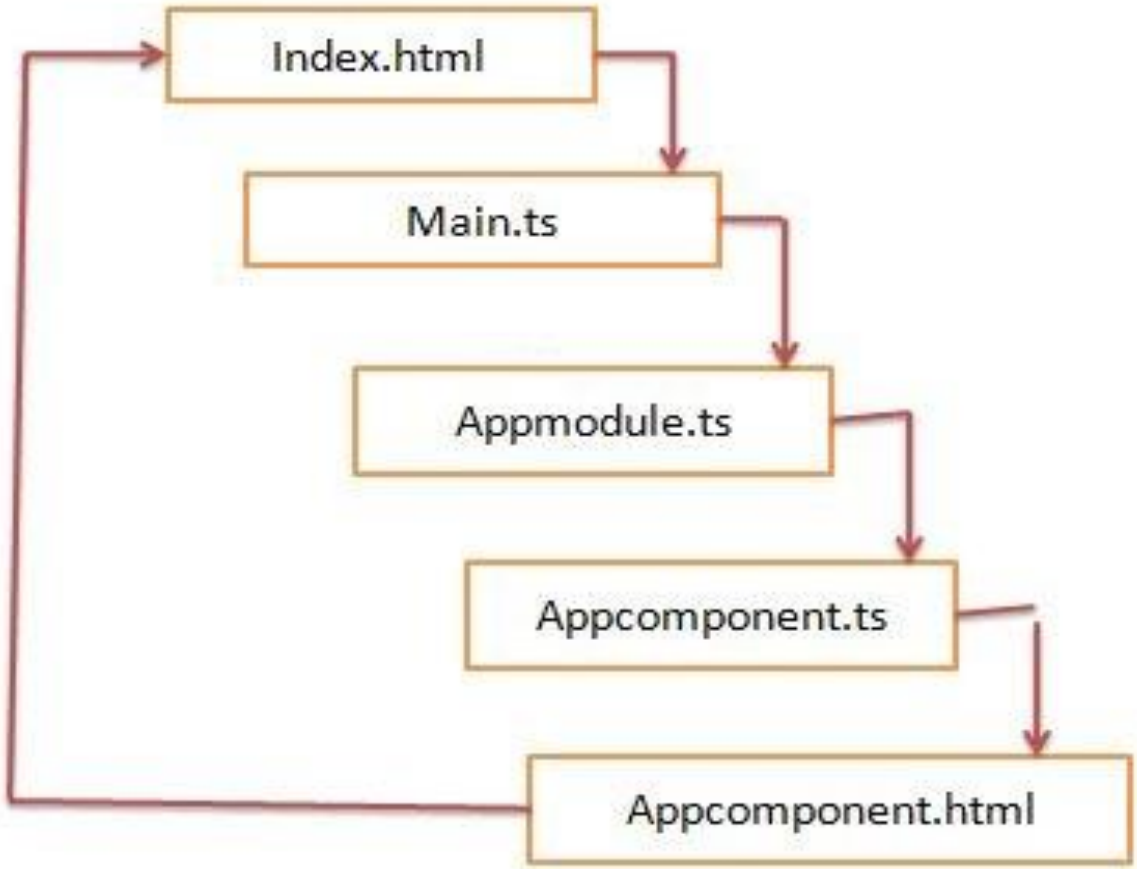
Installing Angular CLI

- To install Angular on your local system, you need the following:
  - **Node.js**
  - **npm package manager**
- The **Angular CLI** is the fastest, straightforward, and recommended way to develop Angular applications.
- The Angular CLI also comes with its **own built-in server**,
  - which we **can use to serve up the Angular application** as we are building it up, and then
  - **view our Angular application in the browser** as a live preview of our application.
  - As when we make changes, the **changes will be reflected immediately to the browser.**
  - **can perform a variety of ongoing development tasks such as testing, bundling, and deployment.**

- To install the Angular CLI, open a terminal window and run the following command:
  - npm install -g @angular/cli
- To create a new Angular Application:
  - ng new my-app
- Run the application
  - ng serve

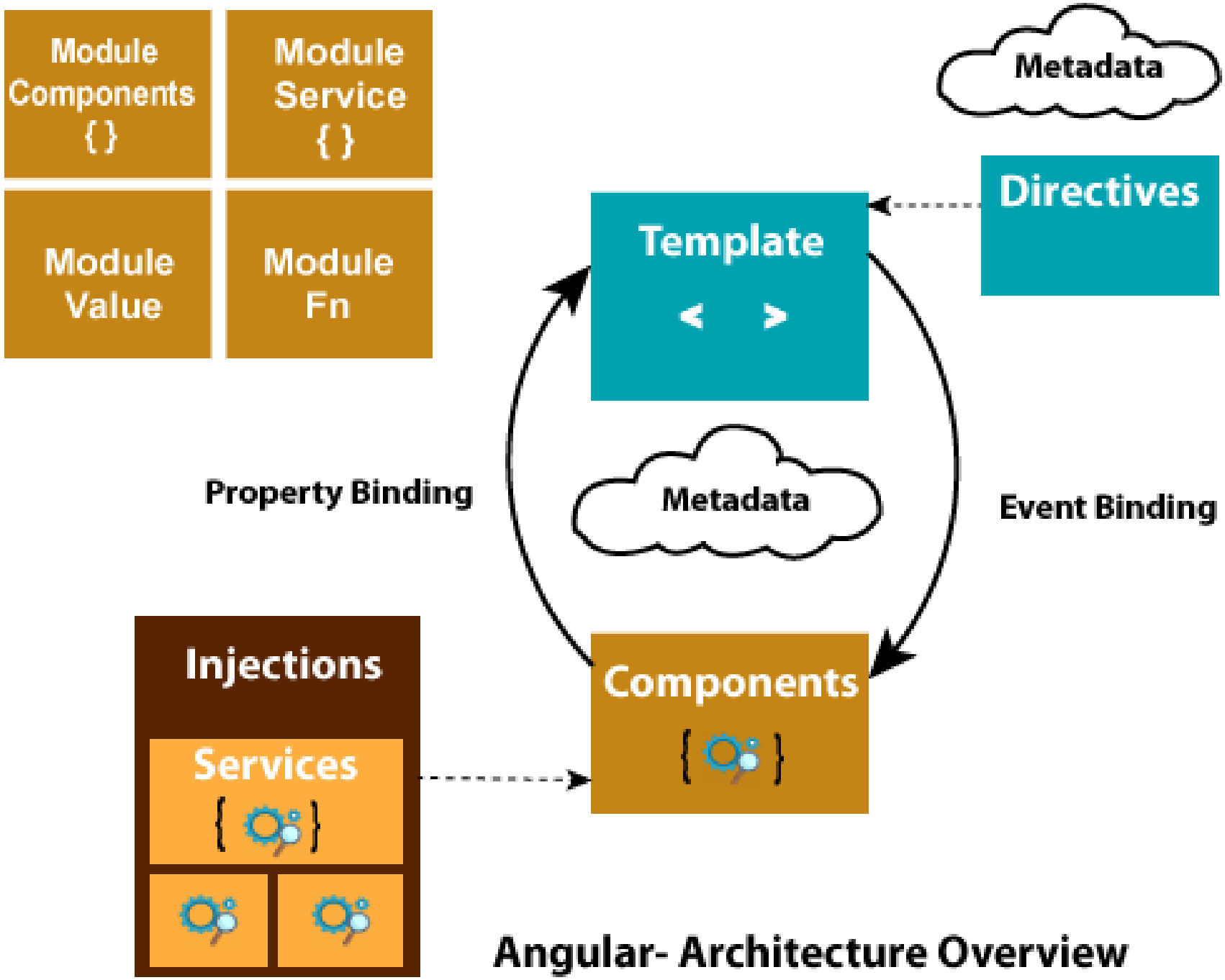
Open the browser and type http://localhost:4200 to the output of the application

Angular Application WorkFlow



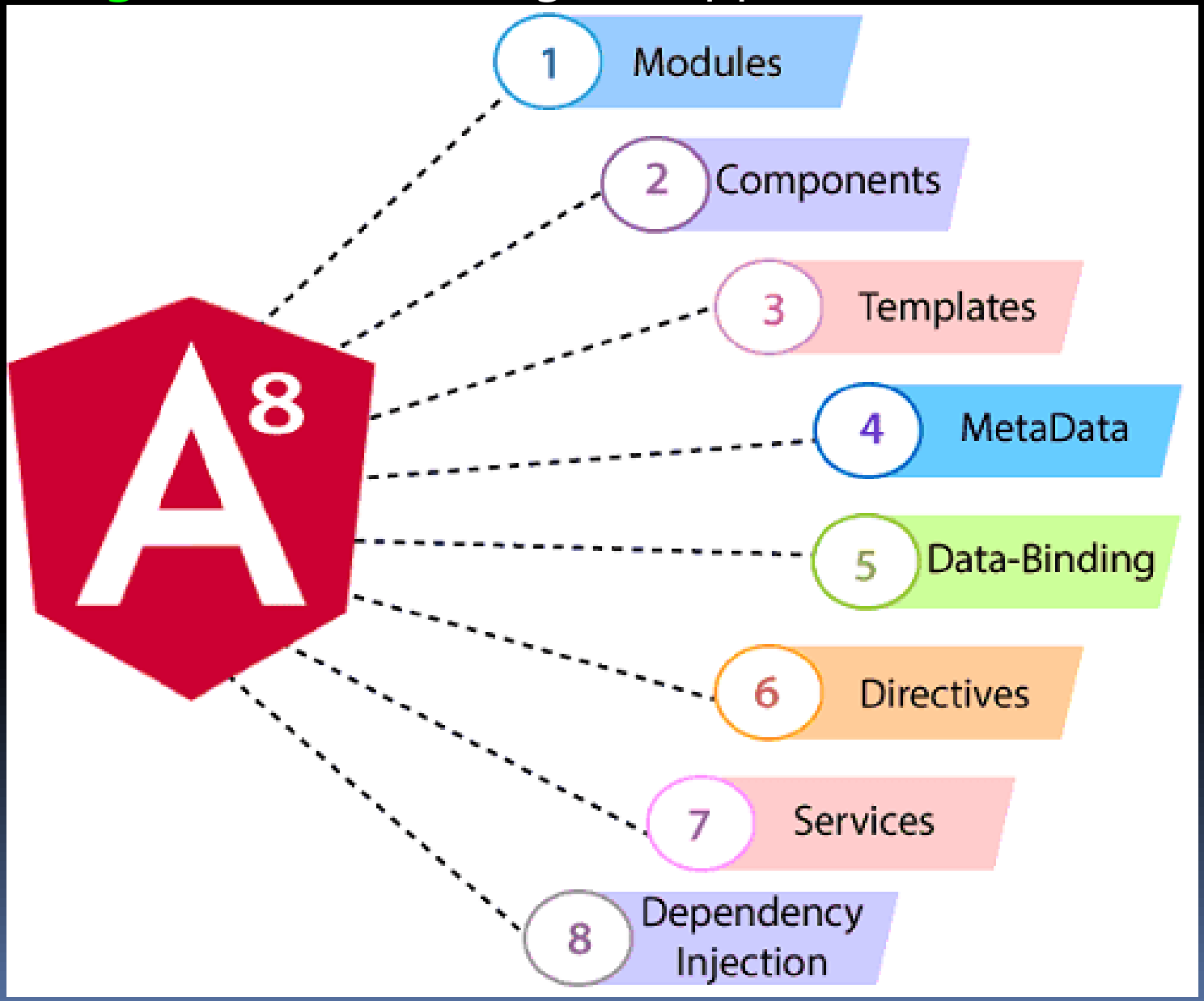
4.7 Angular Architecture

- Angular is a framework for building client applications as a combination of **HTML** and **TypeScript**.
- The framework consists of several libraries, some of them **core** and some optional.
- Angular application is modular in its nature and will **consist of several components**, together with their **templates and metadata**, that comprise the application.
- These components, and other parts of the Angular application, like services, will be organized into **modules**.


Module Components { }
Module Service { }
Module Value
Module Fn
Metadata
Directives
Template
Property Binding
Metadata
Event Binding
Injections
Services
Components
Angular- Architecture Overview

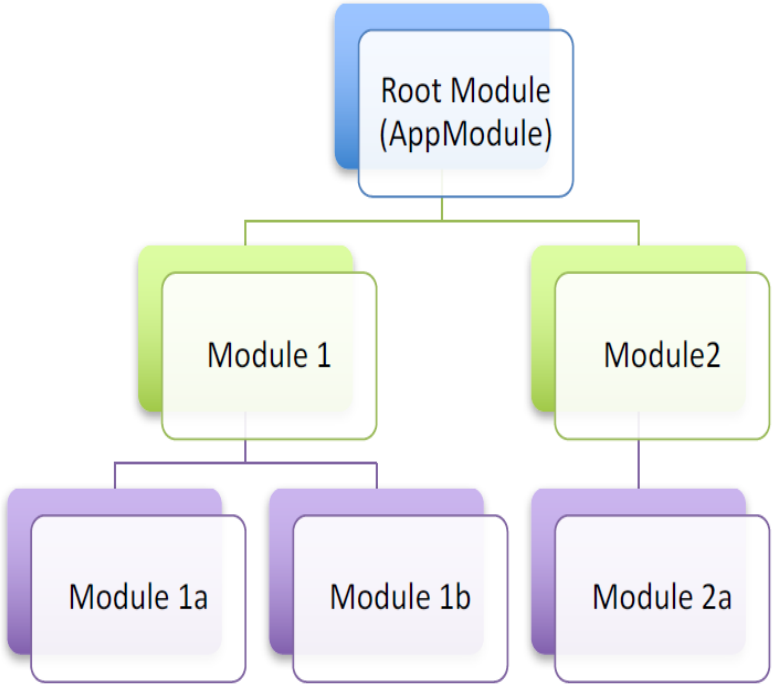
- Angular Applications were developed by:
  - **composing HTML *templates* with Angularized markup**
  - **writing *component* classes to manage those templates**
  - **adding application logic in *services and***
  - **boxing components and services in *modules***
  - **launch the app by *bootstrapping* the *root module*.**

- The architecture diagram identifies the **eight main building blocks** of an Angular application:



1. Modules

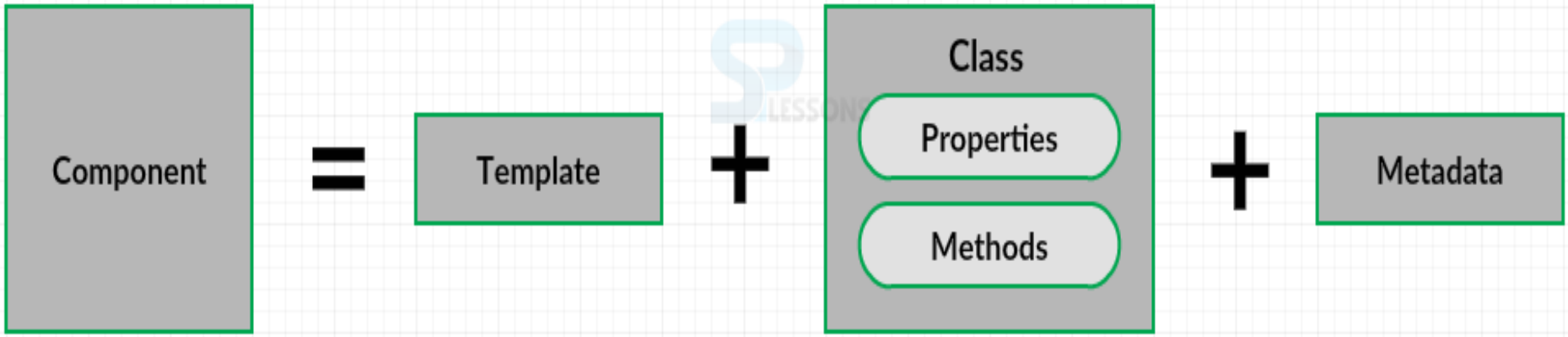
- Angular apps are modular and Angular has its own modularity system called ***Angular modules*** or ***NgModules***.
- Angular apps have one or additional modules.
- "**Module is a typescript class - that has a block of code which is intended to perform a single task**, it is an independent task".
- Every Angular app has at least one Angular module class, the ***root module***, conventionally named **AppModule** and other ***feature modules***
- AppModule/Root Module - **provides the bootstrap mechanism that launches the application.**



- Each module is a collection of related components and services
- Angular modules can export a value/function/class that the other modules can import and make use of.
- An Angular module, whether a *root* or *feature*, is a class with an **@NgModule** decorator.
- *NgModule is a decorator function that takes a single metadata object whose properties describe the module*.
- The most important properties are:
  - **declarations** - the view classes that belong to this module. Angular has three kinds of view classes: components, directives, and pipes.
  - **exports**
  - **imports**
  - **providers -** it's a creator of services. They'll be accessible in all the parts of the application.
  - **bootstrap** - the main application view, called the **root component**, that hosts all other app views. Only the root module should set this bootstrap property.

2. Components

- Components are the main building blocks of an UI in an Angular application and are organized into modules.



- Component is a typescript class with a metadata attached to it.
  - **Class:** contains the core of business logic for the application.
  - **Templates:** used to render the view for the application. This contains the HTML that needs to be rendered in the application. This part also includes the binding and directives.
  - **metadata:** metadata for a component class associates it with a template that defines a view. It is defined with a decorator.

- An angular application can have any number of components.
- At the top of the hierarchy there is a root component.
- Every component consists of @Component decorator associated with it.
- @Component decorator allows us **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.
  - **Properties:**
    - **Selector**
    - **template**
    - **templateUrl**
    - **styleUrls**

3. Templates

- A template is a form of HTML together with angular markup that tells Angular how to render/display the component.
- example:
-

  product name : { {name} }

- Put the template expression inside the interpolation braces in order to display the value.

4. Metadata

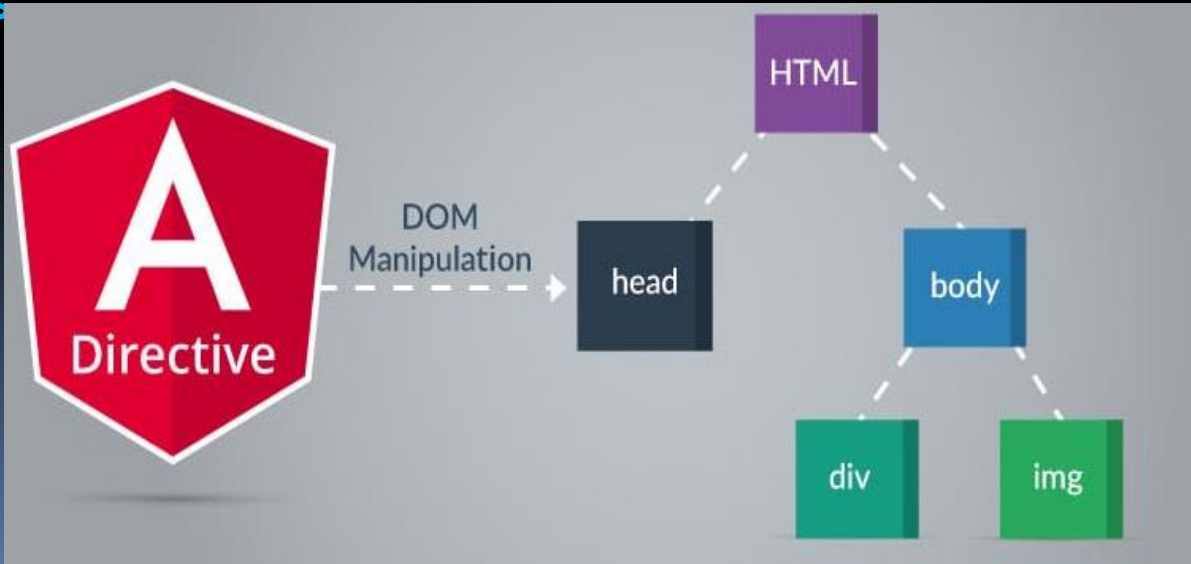
- Metadata **tells Angular how to process a class**.
- In TypeScript, you attach metadata by using a ***decorator***.
- <u>Different Decorators:</u>
  - **@NgModule decorator - Module**
  - **@Component decorator - Component**
  - **@Injectable decorator - Services**
  - **@Directive decorator - Directives**

5. Data Binding

- Angular supports data binding, a mechanism for coordinating parts of a template with parts of a component.
- **Data binding** automatically keeps your page up-to-date based on your application's state.
- It allows us to have communication between a component and a template which is very much necessary to render our business logic to the user.
- Data binding deals with how to bind your data from component to HTML DOM elements (Templates).
- types of bindings that angular supports:
  - Interpolation
  - Property binding
  - Event binding
  - Two-way binding

6. Directives

- They are **extended HTML attributes**.
- Angular templates are *dynamic*. When Angular renders them, it transforms the DOM according to the instructions given by **directives**.
- By using Angular directives, you can change the appearance, behavior or a layout of a DOM/HTML element.
- *Angular Directive* is basically a class with a ***@Directive*** decorator.
- 3 types of Directives:
  - **Component Directives**
  - **Structural Directives**
  - **Attribute Directives**



7. Services

- *Service* is a broad category encompassing any value, function, class or feature that your application needs.
- It is an injectable class which is used to share data among various classes or application.
- It is also responsible to make the server call and get data to display.
- An injector maintains a container of service instances.
- Example services:
  - Logging service, Data service, Application configuration
- Any class annotated with @Injectable and making server calls can be considered as service.
- Example:

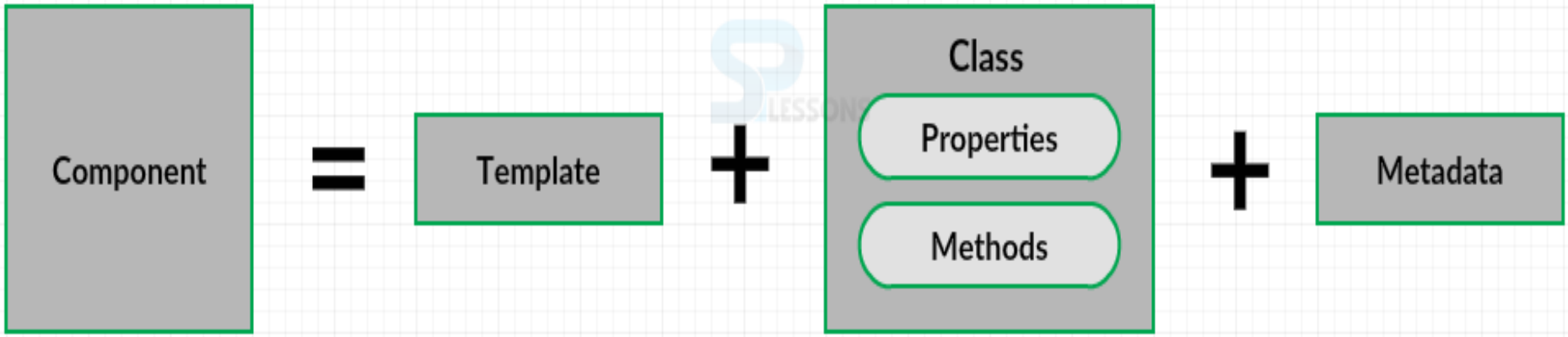
```
@Injectable()
export class BookService {}
```

8. Dependency Injection

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

4.8 Angular Components

- Components are the main building blocks of an UI in an Angular application and are organized into modules.



- Component is a typescript class with a metadata attached to it.
  - **Class:** contains the core of business logic for the application.
  - **Templates:** used to render the view for the application. This contains the **HTML** that needs to be rendered in the application. This part also includes the **binding** and **directive**s.
  - **metadata:** metadata for a component class associates it with a template that defines a view. It is defined with a decorator.

- An angular application can have any number of components.
- At the top of the hierarchy there is a root component also called as AppComponent
- Every component consists of @Component decorator associated with it.
- @Component decorator allows us **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.
  - **Properties:**
    - **Selector**
    - **template**
    - **templateUrl**
    - **styleUrls**

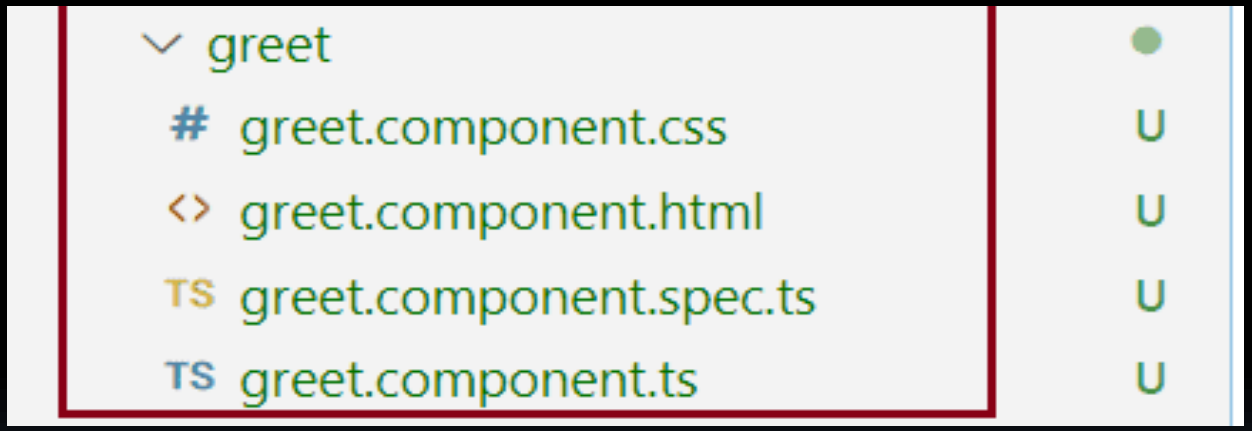
Creating an Angular Component

- STEP 1: To create a component using the Angular CLI:
  - From a terminal window, navigate to the directory containing your application.
  - Use the following CLI command to generate a component.
    - ng generate component <component- name>
    - ng g c <component-name>
- By default, this command creates the following:
  - A folder named after the component
  - A component typescript class file, <component-name>.component.ts
  - A template file, <component-name>.component.html where we will write HTML for a component
  - A CSS file, <component-name>.component.css
  - A testing specification file, <component-name>.component.spec.ts where we can write unit tests for a component

Where <component-name> is the name of your component.

- Example:
  - ng g c greet

  The above command will create a new folder "greet" and creates four files under it, as shown below.



greet.component.ts

Import

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

Metadata

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

Component Tag

HTML Template File Name and Location

CSS File Name and Location

Component Class

```
export class GreetComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }

}
```

- **STEP 2:** Declare a component in the root module.

app.module.ts

```
@NgModule({
        declarations : [GreetComponent]
})
```

- **STEP 3:** Add component tag in the root HTML template. (app.component.html)

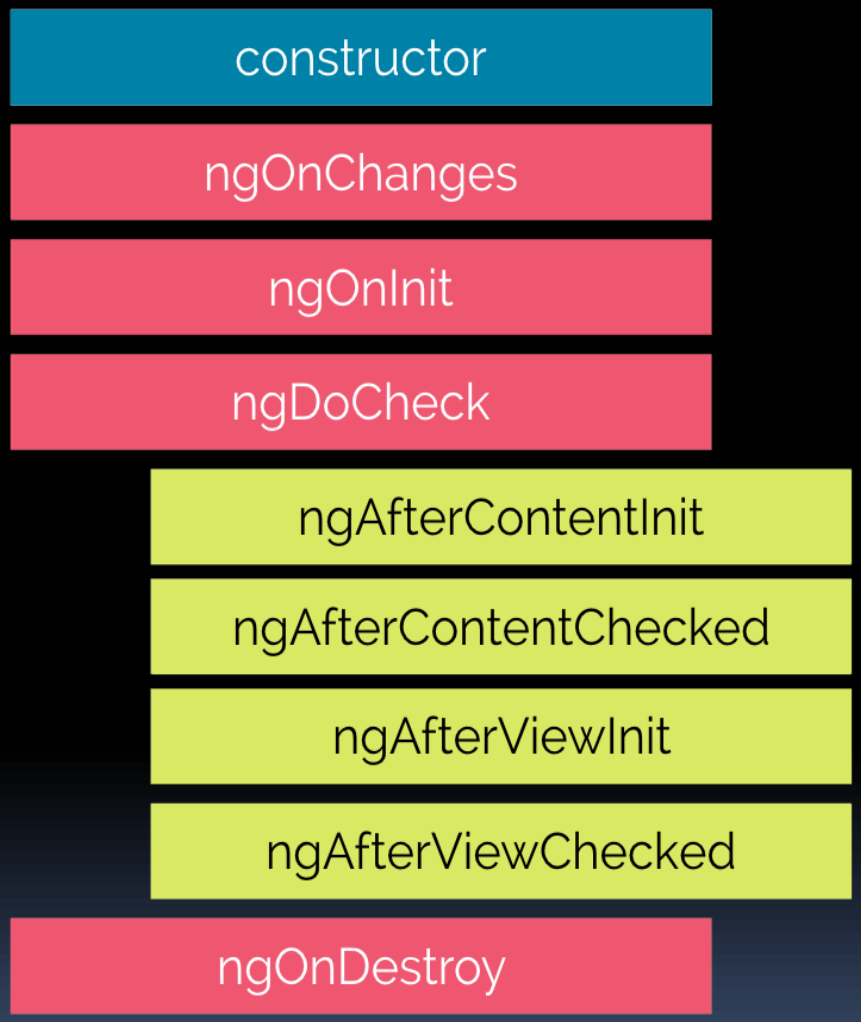
```
<greet> </greet>
```

4.9 Component Life Cycle

- A component has a lifecycle managed by Angular.
- All Components and directives have a lifecycle as Angular creates, updates, and destroys them.
- Life cycle indicates various phases a component follows from start to end.
- Process:
  - A component instance has a lifecycle that starts when Angular instantiates the component class and
  - Renders the component view (html file) along with its children.
  - The lifecycle continues with change detection, as Angular checks to see when data-bound properties change, and updates both the view and the component instance as needed.
  - The lifecycle ends when Angular destroys the component instance and removes its rendered template

Component Life Cycle hooks

- There are 8 different stages in the component lifecycle. Every stage is called **"life cycle hook events"**.
- *After* creating a component/directive by calling its constructor, Angular calls the lifecycle hook methods in the following sequence at specific moments.
- Application can use **lifecycle hook methods** to tap into key events in the lifecycle of a component
- **Lifecycle hook methods are callback functions that Angular calls when a particular event happens.**



Responding to lifecycle events

- Respond to events in the lifecycle of a component or directive by implementing one or more of the *lifecycle hook* interfaces in the Angular core library **@angular/core**.
- "Each interface defines the prototype for a single hook method, whose name is the interface name prefixed with ng."
- For example, the **OnInit** interface has a hook method named **ngOnInit()**

Angular calls these hook methods in the following order:

- ngOnChanges: right at the start when a new component is created and also when a data-binding input bound property changes.
  - Input bound properties are those with **@Input()** decorator
  - Example: @Input() msg: string
- ngOnInit: Called once to initialize the component after the first ngOnChanges.
- ngDoCheck: called during every change detection cycle.
  - Change detection cycle – mechanism using which angular keeps the template in sync with the component class
- ngAfterContentInit: (**only once**) After component projected content (external content projected from the parent component to the child component) is initialized.
- ngAfterContentChecked: After every check of component projected content.

- ngAfterViewInit: After a component's view and all its child views are initialized.
  - View – template of a current component and all of its child components or directives
- ngAfterViewChecked: After every check/update of a component's views and child's views. Fired after ngAfterViewInit and after that during every change detection cycle.
- ngOnDestroy: Just before the component/directive is destroyed by the angular.

4.10 Data Binding in Angular

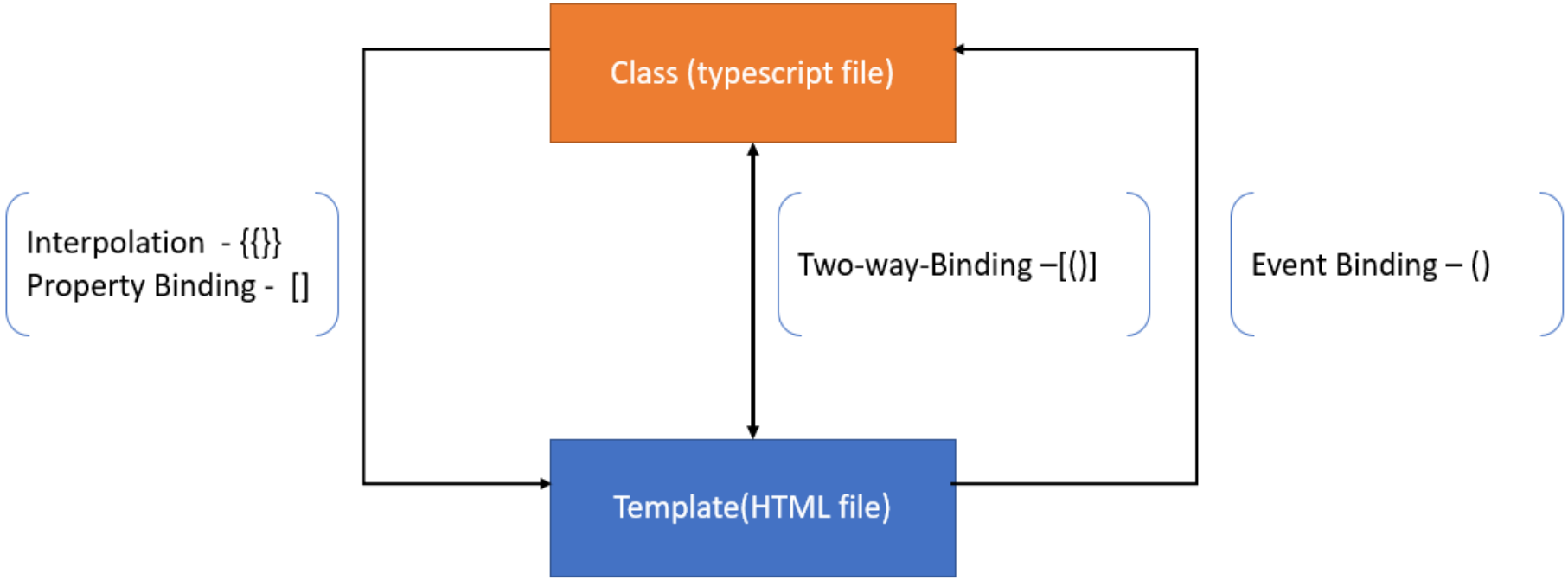
- Angular supports data binding, a mechanism for coordinating **parts of a template** with **parts of a component**.
- **Data binding** automatically keeps your page up-to-date based on your application's state.
- Data binding deals with how to **bind your data** from component to HTML DOM elements (Templates) which is very much necessary to render our business logic to the user.
- Angular allows both **One-way** and **Two-way** Data Binding approaches.

- **One-way binding:**
  - In one-way binding, the **data flow is unidirectional**.
  - **flow of data is from Component (typescript file) to view (HTML file) OR from view to the component**
  - when we make changes in TypeScript code, the HTML template is changed
- **Two-way binding:**
  - In a two-way binding, the **data flow is bi-directional**.
  - Any changes made in the model part will sync in view part as well as any changes made in view part is synced in model part also.
  - This means that the **flow of code is from Typescript file to HTML template file as well as from HTML template file to Typescript file.**

Types of Data Binding:

- 4 types of bindings that angular supports:
  - Interpolation
  - Property binding
  - Event binding
  - Two-way binding

One-way binding



1. Interpolation binding/ String Interpolation:

- String Interpolation is a **one-way data-binding** technique
- It is used to **transfer the data from a TypeScript code (Component) to an HTML template (view).**
- It uses the **template expression** in ***double curly braces*** to display the data from the component to the view.
- It returns the result of the expression as a **string** to view.
- **Syntax:**
- {{**expression**}}

2. Property binding

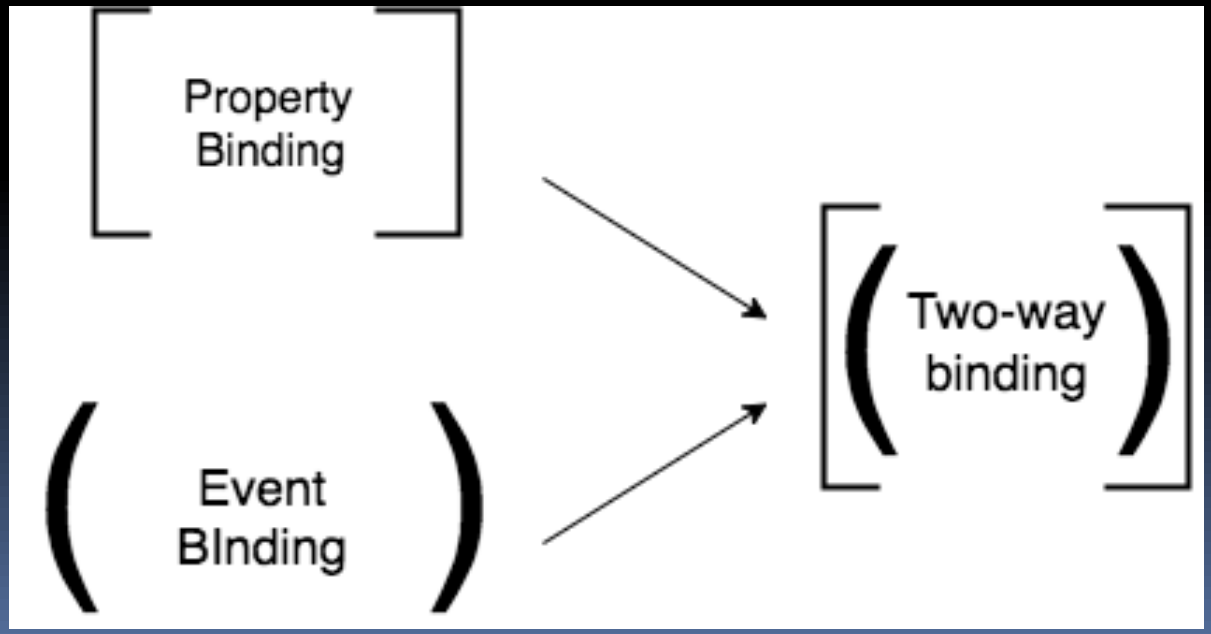
- It is a **one-way data-binding** technique
- allows you to **set the properties for HTML elements/tags**.
- It is used to **bind values of component/model properties to the HTML element/tag properties**.
- **Syntax:**
- **[targetproperty]="expression"**

3. Event binding

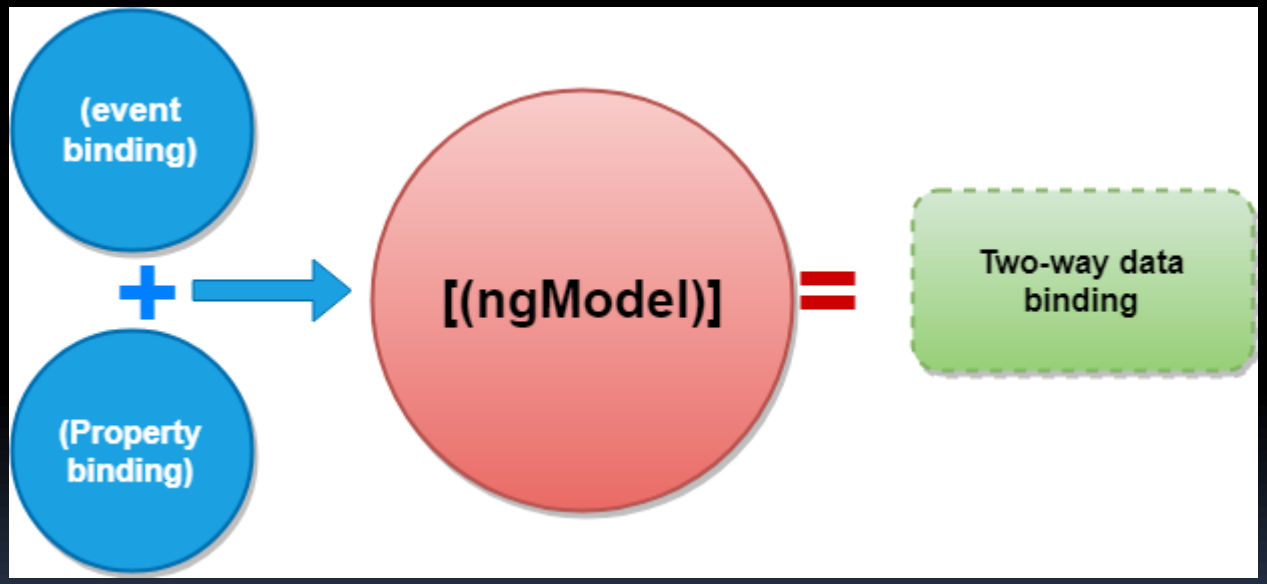
- It is a **one-way data-binding** technique in which flow is from the **view to the component**
- event binding is **used to handle the events raised from the HTML elements in Templates** like button click, mouse move etc.
- When the event happens it **calls the specified method in the component** to handle that event.
- **Syntax:**
- **(target_event_name) = "template_statement"**

4. Two-way binding

- In a two-way binding, the data flow is bi-directional.
- Two-way binding = event binding + property binding
- Any changes to the view/HTML are propagated to the component class. Also, any changes to the properties in the component class are reflected in the view/HTML.
- Angular's two-way binding syntax is a combination of square brackets and parentheses, [()]



- In order to achieve a two-way binding, we will **use ngModel** or **banana in a box** syntax.
- **declare the ngModel directive and set it equal to the name of the property**.

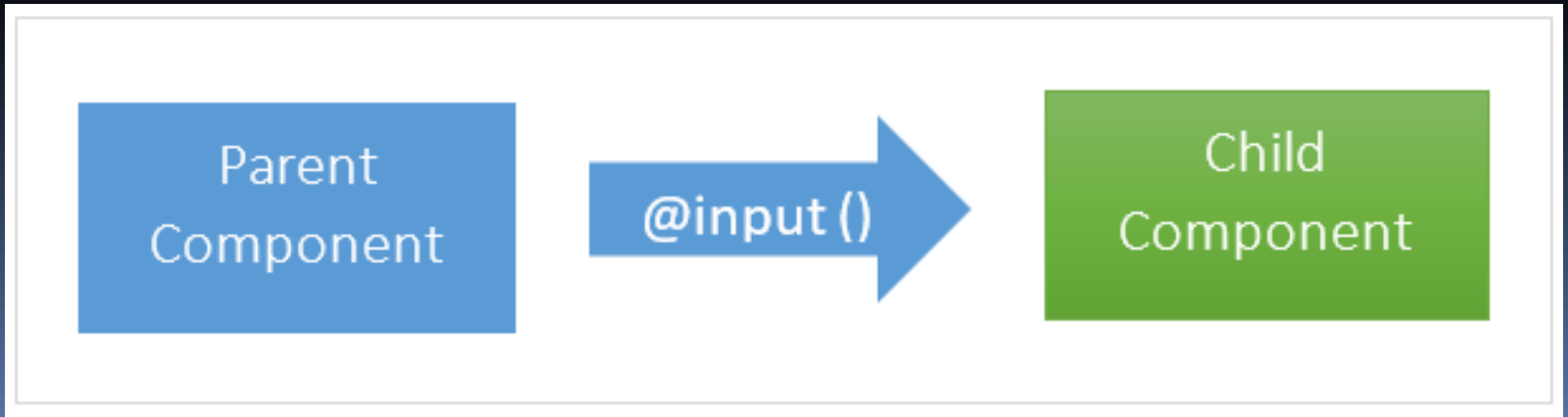


4.11 Component Communication

- There are different ways in which components can communicate or share data between them.
- These approaches depend on whether the components have a Parent-child relationship between them are not.
- There are three Possible approaches:
  - Parent to Child Communication
  - Child to Parent Communication
  - Interaction when there is no parent-child relation

Parent to Child Communication

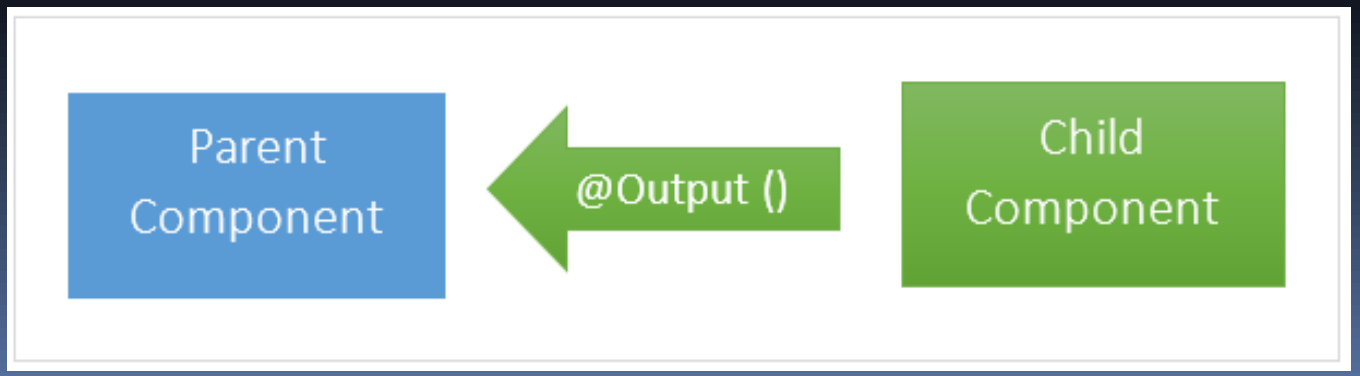
- If the Components have a parent-child relationship then, then the parent component can pass the data to the child by using the **@Input decorator in Angular**
- **Syntax:**
  - @Input () [property-name] : [property-data-type]
- **Procedure:**
  - Create a property in the Child Component and decorate it with @Input().
  - And in the Parent Component Instantiate the Child Component. Pass the value to the Property using the Property Binding Syntax



Child to Parent Communication

- **@Output decorator** is used to pass the data from child to parent component.
- The **child component uses the @Output() property to raise an event of type EventEmitter** to notify the parent of the change.
- Capture event from child component in the parent component.
- emit data from the child component through an event which can be received by the parent component.
- Syntax:

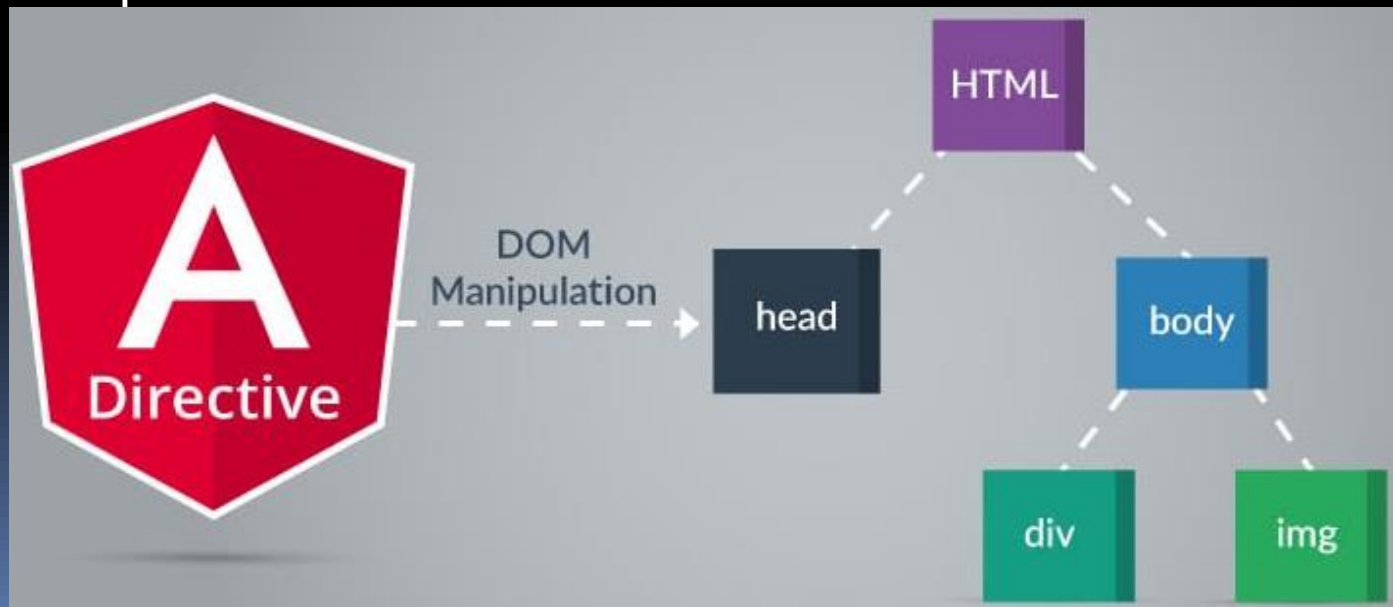
```
@Output () property-name = new EventEmitter ()
```



- Sharing data between components, When there is no relationship between components is done using **Services.**

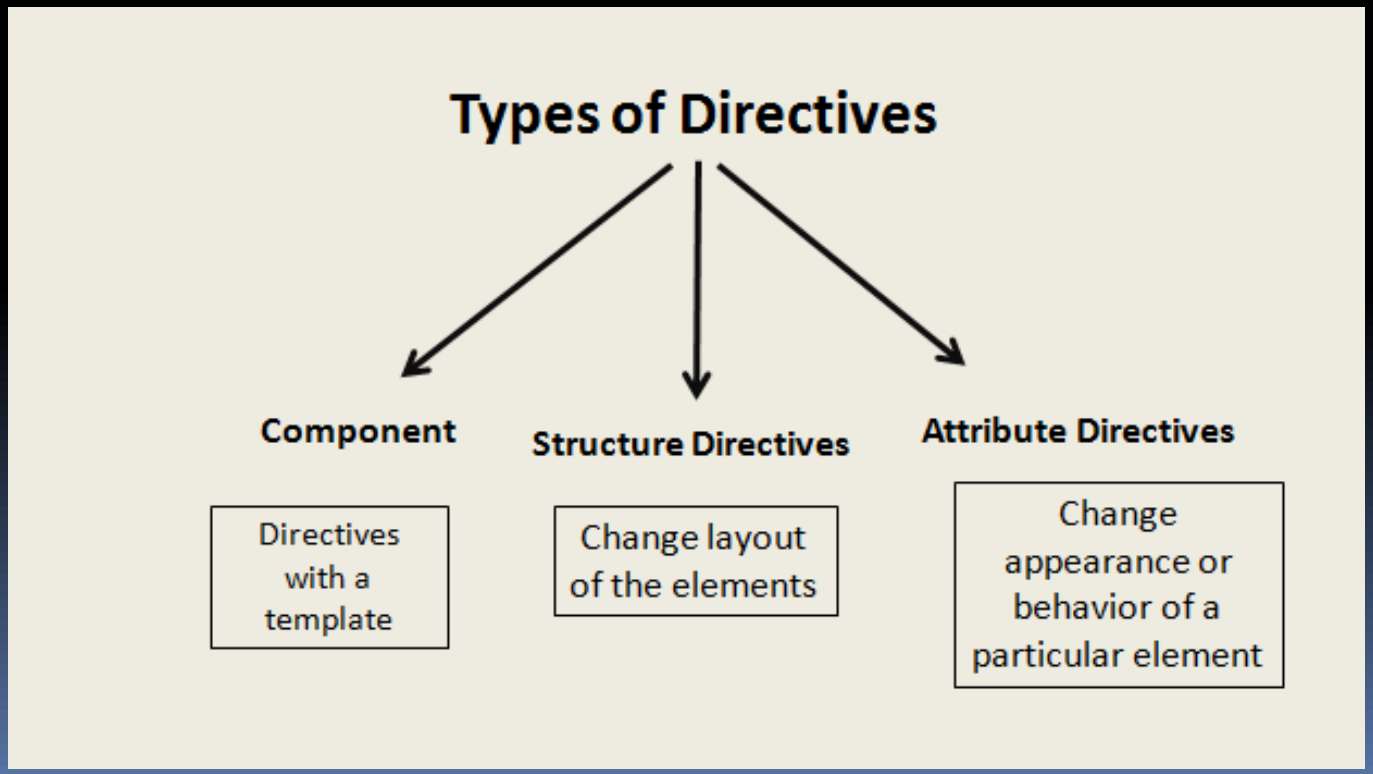
4.12 Angular Directives

- They are **extended HTML attributes** with the prefix "**ng** "
- By using Angular directives, you can change the appearance, behavior or a layout of a DOM/HTML elements.
- Angular templates are *dynamic*. Directives give instructions to Angular on how to render the templates/HTML tags.
- Directives are classes that add additional behavior to elements in your Angular applications.
- They are Typescript classes which are declared with decorator **@Directive.**



Types of Directives

- Components—**directives with a template**. **This type of directive is the most common directive type.**
- Structural directives—directives that change the DOM layout by adding and removing DOM elements.
- Attribute directives—directives that change the appearance or behavior of an element, component, or another directive.



Component Directives:

- It forms the main component class and is declared by **@Component decorator**.
- It contains the details on component processing, instantiated and usage at run time.
- Syntax:

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

Structural Directives

- Structural directives are responsible for HTML layout.
- Allows you to alter the layout by adding, removing and replacing elements in DOM.
- They have a * sign before the directive.
- Commonly used Structural Directives:
  - *ngIf
  - *ngFor
  - ngSwitch

- ***ngIf:**
  - used to display or hide the DOM Element based on the expression value assigned to it.
  - The expression value may be either true or false
  - Syntax:

    <div *ngIf="condition">Content to render when condition is true.
- *ngIf-else:
  - When the expression evaluates to true, Angular renders the template provided in a **then** clause, and
  - when false or null, Angular renders the template provided in an **optional else** clause.

- **Syntax:**
  - <div ***ngIf="condition; else elseBlock"**>Content to render when condition is true.

    <ng-template **#elseBlock**>Content to render when condition is false.</ng-template>
  - <div ***ngIf="condition; then thenBlock else elseBlock"**>

    <ng-template **#thenBlock**>Content to render when condition is true.</ng-template>

    <ng-template **#elseBlock**>Content to render when condition is false.</ng-template>

- *ngFor:
  - used to loop through the dynamic lists in the DOM.
  - **Syntax:**
    - <div ***ngFor="let item of item-list"**>

- **ngSwitch** is a set of three directives:
  - ngSwitch
  - *ngSwitchCase
  - *ngSwitchDefault
- ngSwitch is used to choose between multiple case statements defined by the expressions inside the *ngSwitchCase and display on the DOM Element according to that.
- If no expression is matched, the default case DOM Element is displayed.
- **Syntax:**

```
<div [ngSwitch]="expression">
        <div *ngSwitchCase="expression_1"></div>
        <div *ngSwitchCase="expression_2"></div>
        <div *ngSwitchDefault></div>
</div>
```

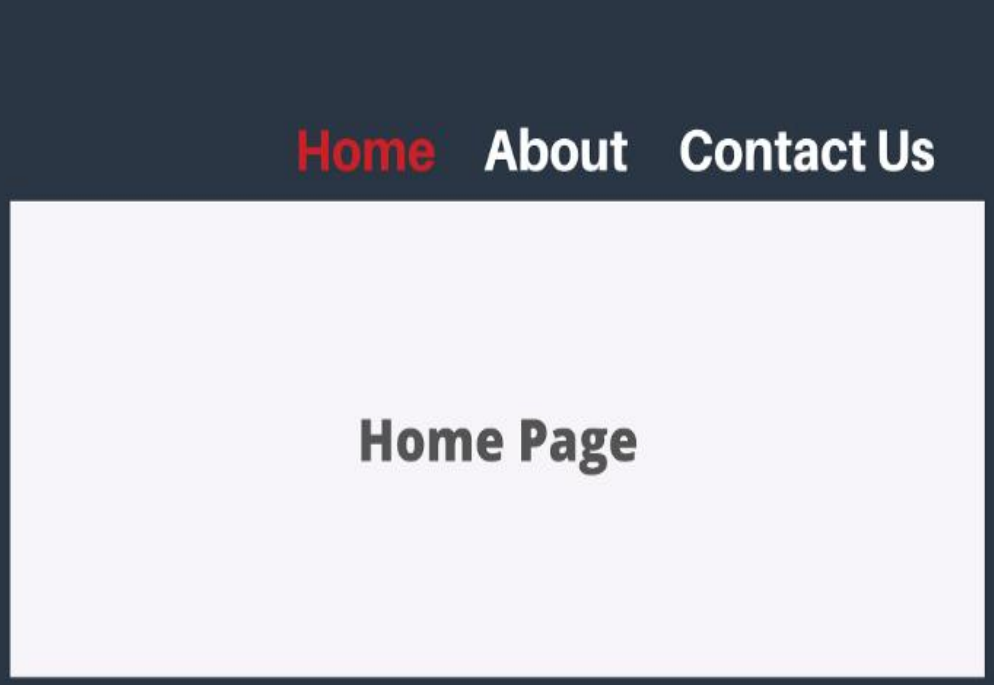
Attribute Directives:

- Attribute directives are **used to change the appearance/look and behavior of the DOM element.**
- It provides the facility to create our own directive.
- Commonly used Attribute Directives:
  - **ngClass**—It controls the appearance of elements by adding and removing CSS classes dynamically.
  - **ngStyle**— Used to apply inline-styles to a HTML element
  - **ngModel**—adds two-way data binding to an HTML form element.

- **ngClass:**
  - <some-element [ngClass]="'first second'">…</some-element>
  - <some-element [ngClass]="{'first': true, 'second': true, 'third': false}">…</some-element>
- **ngStyle:**
  - <some-element [ngStyle]="{'property': styleExp}"> …</some-element>
- **ngModel:**
  - <some-element **[(ngModel)] = "source">**…</some-element>

4.13 Angular Routing

- In a Single Page Application (SPA), all of your application's functions exist in a single HTML page.



- As users access your application's features, they need to move from one view to another view that corresponds to particular components in an application.
- Angular provides a separate module, the **Router module**, for setting up navigation in an Angular application.
- The Router enables navigation by interpreting a browser URL as an instruction to change the view.
- Steps:
  1. Generate an application with routing enabled

     ng new routing-appname -- routing
  2. Add components for routing.

     ng generate component component-name
  3. Import your new components into AppRoutingModule file

4. Define routes using RouterModule and Routes:

- Import RouterModule and Routes into your routing module.
- Define your routes in your Routes array.
- Routes array contains all possible routes for an application
- Each route is an object in Routes array that contains **path** which is reflected in the URL and the **component** to be rendered when we navigate to that path

```
const routes: Routes = [
  { path: 'name', component: ComponentName },
];
```

- 5. Access Routes using router-outlet tag and routerLink or href, property
- Include router-outlet directive in the root component template.

```
<router-outlet></router-outlet>
```

- Use routerLink directives in the required place.

```
<a routerLink="path"  or href ="path" >Text</a>
```

  - routerLink set the route to be called using the path.