<ng-template>

Definition

- Angular <ng-template> defines a template which is not rendered by default.
- We have to explicity tell angular that we want to render this template.
- So we can say <ng-template> is lazy.
- <ng-template> can use properties from its parent context. It can also have its own private context as we can pass values in the ngTemplate and use inside it. This makes it a disconnected and standalone piece which can be reused.
- <ng-template> elements are represented as instances of the TemplateRef class.
- <ng-template> is consumed by diagnostic comments in finished DOM.

Usage

- One of the main usage is to hold template content that will be used by structural directives. *ngIf behind the scene uses <ng-template>
  - <p *ngIf="condition> Content is syntatic sugar for: <ng-template [ngIf]="condition> Content </ng-template>
- <ng-template> elements can be referenced in templates using standard template variables.
  This is how <ng-template> elements are used as ngIf else clauses.
  - <div *ngIf="condition;else error">
    Data

    <ng-template #error>
    <h1>Error</h1>
    </ng-template>
  - Such template variables can be used in conjunction with ngTemplateOutlet directives to render the content defined inside <ng-template> tags.
    - <ng-template #user let-name="username" let-hobbies="userHobbies">
      <h1>{{name}}</h1>

      <li *ngFor="let hobbie of hobbies">
      {{hobbie}}

      </ng-template>
      <div *ngTemplateOutlet="user;context:{username:'Sanjay',userHobbies:['Coding','Workout']}">

      - Defining variables in ng-templates is bit different we have to use something like let-name instead of let name
- A Query (such as ViewChild) can find the TemplateRef associated to an <ng-template> element so that it can be used programmatically; for instance, to pass it to the ViewContainerRef method createEmbeddedView().
  - abstract createEmbeddedView<C>(templateRef: TemplateRef<C>, context?: C, index?: number):
  - This is for directive where we get TemplateRef and ViewContainerRef via Dependency Injection:
    - constructor(
      private viewContainerRef:ViewContainerRef,
      private templateRef:TemplateRef<any>
) { } this.viewContainerRef.createEmbeddedView(this.templateRef);
    - We can use ViewChild query in Component to get TemplateRef and ViewContainerRef reference
- Inside the <ng-template> tags you can reference variables present in the surrounding outer template. Additionally, a context object can be associated with <ng-template> elements. Such an object contains variables that can be accessed from within the template contents via template (let and as) declarations.
  - Template Way
    - <ng-template #user let-name="username" let-hobbies="userHobbies">
      <h1>{{name}}</h1>

      <li *ngFor="let hobbie of hobbies">
      {{hobbie}}

      </ng-template>
      <div *ngTemplateOutlet="user;context:{username:'Sanjay',userHobbies:['Coding','Workout']}">

  - Component Pragmmatic way:
    - Same can be done via ViewContainerRef
- We can pass ng-template (TemplateRef) as Input values to Components and then use *ngTemplateOutlet to provide context thus making code more reusable
  - Parent Component
    - 1. <ng-template #user let-name="username" let-hobbies="userHobbies">
      <h1>{{name}}</h1>

      <li *ngFor="let hobbie of hobbies">
      {{hobbie}}

      </ng-template>
    - <app-pass-ng-template-via-input-binding [userTemplate]="user" ></app-pass-ng-template-via-input-binding>
  - Child Component
    - @Input() userTemplate:TemplateRef<any>|undefined;
    -
      <ng-template *ngTemplateOutlet="userTemplate!;context:{username:'Sanjay',userHobbies:['Coding','Workout']}">
      </ng-template>