Angular Lab 3: Directives Exercise Solutions

1 Lab setup

2 Generating a new Angular project

3 NgClass

Q1: Create a <div> element and nest a <p> with some text within this <div>.

Q2: The CSS Box model provides a set of properties to delineate an element and its child elements. You can use a variety of properties to control the border look for the <div>. Add 2 different classes in the template CSS which sets different values for the `border-style` property, another 2 different classes which sets different values for the `border-color` property, and another 2 different classes which sets different values for the `border-width` property. Give these classes any suitable names. Set a suitable `padding` for the <div>.

Q3: Add any one of the 6 classes to the <div>, and use a string to add another 2 more of these classes to the <div>

Q4: Add any one of the 6 classes to the <div>, and use an object to add another 2 more of these classes to the <div> while removing the existing class.

Q5. Add any one of the 6 classes to the <div>, and use an array of strings to add another 2 more of these classes to the <div>

app.component.ts

```
// Q3
borderSingleString = "thickborder hotborder";

//Q4
borderClassesToAdd = {
  stylishborder : false,
  thickborder : true,
  boringborder : true,
}

//Q5
borderClassesInArray = ['thinborder','freezingborder'];
```

app.component.html

```
<!-- Q1 -->
<div>
  <p>Some random text inside this first paragraph </p>
</div>

<!-- Q3 -->
<div class="stylishborder" [ngClass]="borderSingleString">
  <p>Some random text inside this first paragraph </p>
</div>

<!-- Q4 -->
<div class="stylishborder" [ngClass]="borderClassesToAdd">
  <p>Some random text inside this first paragraph </p>
</div>

<!-- Q5 -->
<div class="stylishborder" [ngClass]="borderClassesInArray">
  <p>Some random text inside this first paragraph </p>
</div>
```

`app.component.css`

```
/* Q2. */

.div {
    padding: 10px;
}

.stylishborder {
    border-style: inset;

}

.boringborder {
    border-style: dotted;
}

.thickborder {
    border-width: 20px;
}

.thinborder {
    border-width: 2px;
}

.hotborder {
    border-color: red;
}

.freezingborder {
    border-color: blue;
}
```

4 NgStyle

Q1: Create a <div> element and nest a <p> with some text within this <div>.

Q2: Create an object whose properties represent CSS properties in the component. The CSS Box model provides a set of properties to delineate an element and its child elements. You can use a variety of properties to control the border look for the <div>. Set a suitable `padding` for the <div>.

Q3: Use NgStyle to apply this object to style the <div> element from Q1

app.component.ts

```
  // Q2
  styleTheBorder = {
    'border-style': 'dotted',
    'border-width': '20px',
    'border-color': 'red'
  }
```

app.component.html

```
<!-- Q1 -->
<div>
    <p>Some random text inside this first paragraph </p>
</div>

<!-- Q3 -->
<div [ngStyle]="styleTheBorder">
    <p>Some random text inside this first paragraph </p>
</div>
```

5 NgIf

Q1: Create a text field and use either event binding to a component method or two-way binding with NgModel to track the value in the text field

Q2. Create a <p> element with some random text within it and apply NgIf to display this element when the number of characters in the text field from Q1 is more than 5.

app.component.ts

```
  // Q1
  randomText: string = "";
```

app.component.html

```
<!-- Q1 -->
<br>
<br>
<label for="randomText">Enter some text :</label><br>
<input type="text" id="randomText" [(ngModel)]="randomText">

<!-- Q2 -->
<p *ngIf="randomText.length > 5">The length of the text is more than 5
characters</p>
```

6 NgFor

Q1: Create a input field for a number and use either event binding to a component method or two-way binding with NgModel to track the value in this field

Q2. Display a list whose items count from 1 up to the value in this field using NgFor. For e.g. if the value is 5, then the list will look like:

- Count : 1
- Count : 2
- Count : 3
- Count : 4
- Count : 5

To accomplish this, we will need an array containing the numbers 1 to the value in this field. This array will have to be dynamically created depending on the value in the input field from Q1. Provide an additional button with event binding to a component method that dynamically create this array.
You can use this shortcut for creating this array dynamically:
https://stackoverflow.com/questions/3746725/how-to-create-an-array-containing-1-n

app.component.ts

```
  // Q1
  randomNumber : number = 0;

  // Q2
  arrayNum : number[] = [];

  generateList() {
    this.arrayNum = Array.from(Array(this.randomNumber).keys());
  }
```

app.component.html

```
<!-- Q1 -->
<br>
<br>
<label for="randomNumber">Enter a number :</label>
<input type="number" id="randomNumber" [(ngModel)]="randomNumber">

<!-- Q2 -->
<br>
<br>
<button type="button" (click)="generateList()">Generate List</button>
<br>

<ul>
  <li *ngFor="let num of arrayNum"> Count : {{  num + 1 }} </li>
</ul>
```

6.1 Accessing index of item in collection

6.2 Styling using first, last, odd and even exported values

Q1: Create another <div> containing a collection of buttons using NgFor. Repeat the styling for the <div> and the buttons using odd, even, first and last but this time using either style binding or NgStyle

app.component.html

```
<hr>

<!-- Q1 Use the ternary operator (more than once if necessary)
  in order to decide the specific style rules to apply
  depending on the exported values from NgFor
-->
<div class="button-container" *ngFor="let emp of employees; let i = index;
let isItOdd = odd; let isItEven =even; let isItFirst = first; let isItLast =
last"
[ngStyle]="{
  'border' : isItFirst ? '4px dotted purple' : isItLast ? '4px dashed red' :
'2px solid blue'
}"
>
```

```
  <button [ngStyle]="{
    'background-color' :  isItOdd ?  'aliceblue' : 'aquamarine'
  }" (click)="showButtonInfo(i)">
    New Button # {{i}} : {{emp.name}}</button>
</div>
```

6.3 NgFor with child components

Q1: Create another domain model class called Order with the following properties:

- description: string;
- quantity: number;
- price: number;
- totalCost: number;

Include a method calculateCost in this class to calculate the totalCost = price * quantity
The constructor for this class should only initialize description, quantity and price. The totalCost value is computed by calling the calculateCost method.

Q2: Create a new child component to receive and display data for objects created from the Order class.

Q3: Create appropriate elements in the child template to display each property from an Order object properly using interpolation, including totalCost.

Q4: Create appropriate @Input properties in the child component to receive the Order objects and number from the parent template

Q5: Create an array of 5 Order objects with random values in the parent component, and calculate the totalCost using the calculateCost method

Q6: In the parent template, using NgFor together with parent-child binding to pass each object as well as the number of each object to the @Input properties in the child component.

`Order.ts`

```
// Q1
export class Order {

    description: string;
    quantity: number;
    price: number;
    totalCost: number = 0;

    constructor(description: string, quantity: number, price: number) {
        this.description = description;
        this.quantity = quantity;
        this.price = price;
    }

    calculateCost() {
        this.totalCost = this.price * this.quantity;
    }

}
```

Q2: Generate component via

`ng generate component Order`

`order.component.ts`

```
import { Component, Input } from '@angular/core';
import { Order } from '../Order';

// Q4

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

  @Input()
  myOrder!: Order;

  @Input()
  orderNum!: number;
}
```

`order.component.html`

```
<!-- Q3 -->

<div>
    <p>Order # {{orderNum+1}}</p>
    <p>Description : {{myOrder.description}}</p>
    <p>Quantity : {{myOrder.quantity}}</p>
    <p>Price : {{myOrder.price}}</p>
    <p>Total Cost : {{myOrder.totalCost}}</p>
</div>
```

order.component.css

```
/* Q3 */
div {
    width: 200px;
    padding: 5px;
    margin: 10px;
    border: 2px solid blue;
}
```

app.component.ts

```
  // Q5
  orders: Order[] = [];

  constructor() {

    // Q5
    this.orders.push(new Order("Car",7,5));
    this.orders.push(new Order("Bicycle",10,20));
    this.orders.push(new Order("Boat",16,30));
    this.orders.push(new Order("Motorbike",20,100));
    this.orders.push(new Order("Airplane",3,250));

    // Determine the total cost
    for (let order of this.orders) {
      order.calculateCost();
    }

  }
```

app.component.html

```html
<!-- Q6 -->
<hr>
<app-order *ngFor="let order of orders; let i=index;" [myOrder]="order"
[orderNum]="i"></app-order>
```

7 Combining NgIf and NgFor

Q1: Using the Solution from 6.3 (Q1 - Q6), add an additional button to flip the value of a Boolean property in the parent component.

Q2: Based on this button, either perform or ignore the ngFor together with parent-child binding that is used to pass each object / index number of object in array to the child component

Q3: Further modify / extend the parent-child binding to only pass Order objects whose totalCost is more than a specific value to the child component. Choose this specific value and the random values for your 5 Order object to demonstrate that only certain Order objects are passed.

app.component.ts

```typescript
// Previous Q5
orders: Order[] = [];

constructor() {

  // Previous Q5
  this.orders.push(new Order("Car",7,5));
  this.orders.push(new Order("Bicycle",10,20));
  this.orders.push(new Order("Boat",16,30));
  this.orders.push(new Order("Motorbike",20,100));
  this.orders.push(new Order("Airplane",3,250));

  // Determine the total cost
  for (let order of this.orders) {
    order.calculateCost();
  }

}

// current Q1
showOrders: boolean = false;
```

```
  buttonMessage: string = 'Show orders';

  // current Q2
  hideShowOrders() {
    if (this.buttonMessage === 'Show orders')
      this.buttonMessage = 'Hide orders';
    else
      this.buttonMessage = 'Show orders';

    this.showOrders = !this.showOrders;
  }

  // current Q3
  costLowerLimit : number = 300;
```

app.component.html

```

<!-- Q1 -->
<button type="button" (click)="hideShowOrders()">{{buttonMessage}}</button>
<br>
<br>

<!-- Q2 -->

<ng-container *ngIf="showOrders">
  <app-order *ngFor="let order of orders; let i=index;" [myOrder]="order"
[orderNum]="i"></app-order>
</ng-container>

<!-- Q3 modify from solution to Q2 above -->

<ng-container *ngIf="showOrders">
  <ng-container *ngFor="let order of orders; let i=index;">
    <app-order *ngIf="order.totalCost > costLowerLimit" [myOrder]="order"
[orderNum]="i"></app-order>
  </ng-container>
</ng-container>
```

8 NgSwitch

Q1. Create three radio buttons with 3 different possible values

Q2: Using NgSwitch, display 3 different <p> contents depending on the particular radio button selected.

`app.component.ts`

```
  // Q1
  selectedCountry = '';

  setSelectedCountry(val: string) {
    this.selectedCountry = val;
  }
```

`app.component.html`

```
<!-- Q1 -->
<hr>
<input #radio1 type="radio" id="malaysia" name="country" value="malaysia"
(click)="setSelectedCountry(radio1.value)">
<label for="malaysia">Malaysia</label><br>
<input #radio2 type="radio" id="spain" name="country" value="spain"
(click)="setSelectedCountry(radio2.value)">
<label for="spain">Spain</label><br>
<input #radio3 type="radio" id="finland" name="country" value="finland"
(click)="setSelectedCountry(radio3.value)">
<label for="finland">Finland</label><br>

<!-- Q2 -->
<div [ngSwitch]="selectedCountry">
  <p *ngSwitchCase="'malaysia'">Thats a hot country !</p>
  <p *ngSwitchCase="'spain'">Thats a temperate country !</p>
  <p *ngSwitchCase="'finland'">Thats a cold country !</p>
</div>
```