INSTITUTO TECNOLÓGICO DE COSTA RICA
CURSO: ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE
PROFESOR: ANDRÉS VÍQUEZ VÍQUEZ
LABORATORIO: KARMA + JASMINE

Construcción de una calculadora

1. Utilizando la terminal de Visual Studio Code, ubíquese en la carpeta del proyecto con el comando **`cd my-app`**

2. Ahora se debe ejecutar la aplicación angular para iniciar el servidor web y abrir la aplicación en un navegador, por lo que debe usar el comando que se muestra a continuación: **`ng serve`**

3. Ya puede usar la aplicación en el navegador mediante la URL: http://localhost:4200

4. Imagine que le solicitan diseñar una calculadora sencilla que realiza las operaciones aritméticas básicas: *suma*, *resta*, *multiplicación* y *división*, por lo que es necesario agregar un nuevo componente al proyecto: **`ng g component calculadora`**

5. Busque el fichero app.component.html y en la línea antes de <!-- Resources --> escriba el siguiente código:

```
<app-calculadora></app-calculadora>
```

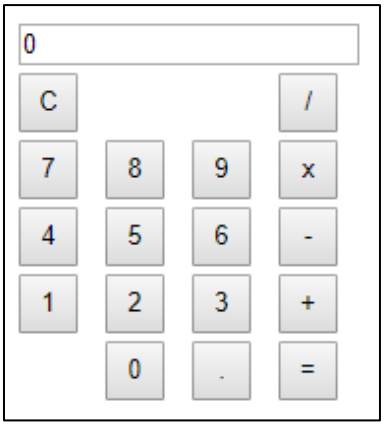
6. La calculadora que se está implementando hará uso de un servicio, por lo que ahora se debe agregar un servicio al componente llamado calculadora. Para esto puede usar la siguiente instrucción **`ng g service calculadora/calculadora`**

7. El código del servicio será el siguiente:

```
import { Injectable } from '@angular/core';
@Injectable({
  providedIn: 'root'
})
export class CalculadoraService {

  constructor() { }

  sumar(numero1: number, numero2: number) {
    return numero1 + numero2;
  }

  restar(numero1: number, numero2: number) {
    return numero1 - numero2;
  }
```

```
  multiplicar(numero1: number, numero2: number) {
    return numero1 * numero2;
  }

  dividir(numero1: number, numero2: number) {
    if (numero2 === 0)
      throw new Error('División por cero');

    return numero1 / numero2;
  }

  calcular(numero1: number, numero2: number, operacion: string) {
    let resultado: number;
    if (numero2 === null) {
      return;
    }

    switch (operacion) {
      case "+":
        resultado = this.sumar(numero1, numero2);
        break;
      case "-":
        resultado = this.restar(numero1, numero2);
        break;
      case "/":
        resultado = this.dividir(numero1, numero2);
        break;
      case "x":
        resultado = this.multiplicar(numero1, numero2);
        break;
      default:
        resultado = 0;
    }

    return resultado;
  }
}
```

8. El siguiente paso es crear un formulario en el archivo calculadora.component.html que tenga la siguiente estructura:



El código sería algo similar al siguiente:

```
<div>
  <form>
    <table>
      <tr>
        <td colspan="4"><input type="text" id="display" name="display"
readonly></td>
      </tr>
      <tr>
        <td><button type="button" id="btnLimpiar">C</button></td>
        <td></td>
        <td></td>
        <td><button type="button" id="btnDividir">/</button></td>
      </tr>
      <tr>
        <td><button type="button" id="btn7">7</button></td>
        <td><button type="button" id="btn8">8</button></td>
        <td><button type="button" id="btn9">9</button></td>
        <td><button type="button" id="btnMultiplicar">x</button></td>
      </tr>
      <tr>
        <td><button type="button" id="btn4">4</button></td>
        <td><button type="button" id="btn5">5</button></td>
        <td><button type="button" id="btn6">6</button></td>
        <td><button type="button" id="btnResta">-</button></td>
      </tr>
      <tr>
        <td><button type="button" id="btn1">1</button></td>
        <td><button type="button" id="btn2">2</button></td>
        <td><button type="button" id="btn3">3</button></td>
        <td><button type="button" id="btnSuma">+</button></td>
      </tr>
      <tr>
        <td></td>
        <td><button type="button" id="btn0">0</button></td>
        <td><button type="button" id="btnDecimal">.</button></td>
        <td><button type="button" id="btnIgual">=</button></td>
      </tr>
    </table>
  </form>
</div>
```

9. Para que todos los botones tengan el mismo tamaño, agrega este código al archivo calculadora.componente.css.

```
button {
    width: 30px;
    height: 30px;
}
```

10. En el archivo calculadora.component.ts importe el servicio.

```
import { CalculadoraService } from './calculadora.service';
```

11. Declare las siguientes variables:

```
private numero1: string = "";
private numero2: any = "";
private resultado: any = 0;
private operacion: any = "";

private calculadoraService:CalculadoraService;
```

12. En el constructor inicialice la variable del servicio.

```
constructor() {
    this.calculadoraService = new CalculadoraService();
}
```

13. Inicie desplegando los datos en el campo de texto, para esto debe agregar el siguiente método:

```
get display(): string {
    if (this.resultado !== null) {
      return this.resultado.toString();
    }
    if (this.numero2 !== null) {
      return this.numero2;
    }
    return this.numero1;
}
```

14. En segundo lugar implemente el botón de limpiar, agregue el método y llámelo en la función ngOnInit.

```
limpiar(): void {
    this.numero1 = '0';
    this.numero2 = null;
    this.resultado = null;
    this.operacion = null;
  }

ngOnInit() {
    this.limpiar();
}
```

15. Actualice las líneas a continuación en el archivo calculadora.component.html

```
<button type="button" id="btnLimpiar" (click)="limpiar()">C</button>
<input type="text" id="display" name="display" [value]="display" readonly>
```

16. Seguidamente, en el archivo calculadora.component.ts, es necesario almacenar los números en las variables, por lo que debe definir los siguientes dos métodos:

```
AdicionarNumero(numero: string): void {
    if (this.operacion === null) {
      this.numero1 = this.concatenarNumero(this.numero1, numero);
    } else {
      this.numero2 = this.concatenarNumero(this.numero2, numero);
    }
  }

concatenarNumero(numAtual: string, numConcat: string): string {
    if (numAtual === '0' || numAtual === null) {
      numAtual = '';
    }
    if (numConcat === '.' && numAtual === '') {
      return '0.';
    }
    if (numConcat === '.' && numAtual.indexOf('.') > -1) {
      return numAtual;
    }
    return numAtual + numConcat;
  }
```

17. En el archivo calculadora.component.html, se debe modificar cada uno de los botones correspondientes a los números y el punto. Este sería el ejemplo para el botón del número 7, recuerde que debe modificar el valor del parámetro en cada botón.

```
<button type="button" id="btn7" (click)="AdicionarNumero('7')">7</button>
```

18. Ahora implemente la funcionalidad de las operaciones, agregando el siguiente método en el documento calculadora.component.ts.

```
definirOperacion(operacion: string): void {
    //Define la operación
    if (this.operacion === null) {
      this.operacion = operacion;
      return;
    }
}
```

19. Al igual que en el caso anterior, en el archivo calculadora.component.html debe modificar cada uno de los botones para llame al método implementado.

```
<button type="button" id="btnDividir"
(click)="definirOperacion('/')">/</button>

<button type="button" id="btnMultiplicar"
(click)="definirOperacion('x')">x</button>
```

```
<button type="button" id="btnResta"
(click)="definirOperacion('-')">-</button>

<button type="button" id="btnSuma"
(click)="definirOperacion('+')">+</button>
```

20. Finalmente, en el archivo calculadora.component.ts, desarrolle el método para calcular el resultado.

```
calcular():void{
    if(this.numero2===null){
      return;
    }

    this.resultado = this.calculadoraService.calcular(
      parseFloat(this.numero1),
      parseFloat(this.numero2),
      this.operacion);
}
```

21. En el archivo calculadora.component.html, en el botón con el signo = debe invocar el siguiente método:

```
<button type="button" id="btnIgual" (click)="calcular()">=</button>
```

Pruebas de servicios

No obstante, de este taller lo que realmente nos interesa es el proceso de pruebas. Así que implementaremos el primer caso de prueba. Para lo que suponga se ha definido un caso de prueba de la siguiente manera.

Caso de prueba	
Código	CP01
Hardware y software	
Sistema operativo	Windows 7
Navegador	
Resolución	1600x900
Comentarios	
Nombre	Suma con valores enteros positivos
Descripción	Sumar dos números enteros positivos
Encargado	Juan Pérez
Precondiciones	
Pasos	Ejecutar la aplicación
	Ingresar los números enteros 10 y 15
	Seleccionar la opción de sumar
Resultados esperados	
Éxito	25

Fracaso	Cualquier número diferente de 25
Prioridad	Alta
Notas	

22. Primero, ubique el archivo calculadora.service.spec.ts, y copie el caso de prueba programado a continuación:

```
import { inject, TestBed } from '@angular/core/testing';

describe('Sumar', function () {
    it('10 + 15 debe ser 25', inject([CalculadoraService], (service:
CalculadoraService) => {
      expect(service.sumar(10, 15)).toBe(25);
    }));
});
```

23. Ejecute el comando **`ng test`** para probar la aplicación de Angular con el marco de pruebas de Jasmine. Karma se ejecuta y muestra 5 casos de prueba ejecutados.

24. Ahora, por su propia cuenta proceda a desarrollar nuevos casos de prueba como para:
    - Sumar dos valores decimales
    - Restar 2 menos 2
    - Restar 3 menos 4
    - Multiplicar 3 por 2

25. Agregue dos nuevos casos de prueba para la división, uno para la división de dos números enteros y otro para la división por cero.

```
describe('Dividir', function () {
    it("2 dividido por 2 debe ser 1", inject([CalculadoraService],
(service: CalculadoraService) => {
      expect(service.dividir(2, 2)).toBe(1);
    }));
    it("6 dividido por 0 debe generar una Excepción",
inject([CalculadoraService], (service: CalculadoraService) => {
      expect(function () { service.dividir(6, 0); }).toThrowError(Error,
'División por cero');
      expect(function () { service.dividir(6, 0);
}).toThrowError('División por cero');
      expect(function () { service.dividir(6, 0); }).toThrowError(Error);
      expect(function () { service.dividir(6, 0);
}).toThrowError(/División por cero/);
    }));
  });
```

26. Verifique nuevamente la ejecución de los casos de prueba para ver los resultados obtenidos.

Pruebas de componentes

27. Recuerde que también se cuenta con una implementación en el componente. Por lo que es necesario hacer pruebas a ese documento, para esto vaya al archivo calculadora.component.spect.ts y agregue los siguientes casos de pruebas para el método concatenar.

```
describe('Cancatenar numero', function () {
    let component: CalculadoraComponent;
      beforeEach(() => {
        component = new CalculadoraComponent();
    });
    it("concatenar 2 + 5 = 25", function() {
      expect(component.concatenarNumero("2","5")).toBe("25");
    });
    it("concatenar Vacío + . = 0.", function() {
      expect(component.concatenarNumero("",".")).toBe("0.");
    });
    it("concatenar Decimal 5. + 5 = 5.5", function() {
      expect(component.concatenarNumero("5.","5")).toBe("5.5");
    });
    it("concatenar Más de un punto 5.5 + . = 5.5", function() {
      expect(component.concatenarNumero("5.5",".")).toBe("5.5");
    });
  });
```

Casos de prueba parametrizables

A menudo es necesario probar un mismo test con diferentes parámetros. En un principio se podría pensar en implementar el mismo test inicializando los valores a testear una y otra vez. Lo mejor es usar tests parametrizables en los que se define el test y sus parámetros de entrada, luego el framework de pruebas se encargará de ejecutar el test para todos los parámetros definidos.
En el siguiente ejemplo, se probará el correcto funcionamiento de un módulo. Se van a implementar una serie de tests y se invocarán con diferentes parámetros para comprobar cómo se ejecuta un mismo test con diferentes datos de entrada.

28. Agregue un nuevo componente llamado Person: **`ng g class Person`**

29. En el archivo person.ts declare una variable global.

```
const AGE_FOR_DRIVE = 18;
```

30. Declare una variable local.

```
private age: number = 0;
```

31. Agregue los métodos set y get para la variable age.

```
/**
   * @return Age of this person
   */
  getAge() {
    return this.age;
  }
  /**
   * @param age Age of this person
   */
  setAge(age: number) {
    this.age = age;
  }
```

32. Implemente un método que valide si una persona puede conducir, considerando que la restricción es que tenga 18 años o más.

```
canDrive() {
    return this.getAge() >= AGE_FOR_DRIVE;
  }
```

33. En el archivo person.spect.ts copie el siguiente código:

```
describe("Pruebas parametrizadas", () => {
      let component: Person;
      beforeEach(() => {
          component = new Person();
});

/** Las pruebas de los proximos pasos se colocan en esta seccion */

});
```

34. El primer caso de prueba consistirá en verificar la correcta inicialización de la variable edad. El siguiente método, que utiliza un parámetro para asignarle un valor a la variable edad mediante el método setAge, debe ser colocado en la sección de comentarios del paso anterior.

```
it('Casos de prueba para validar que la edad es asignada correctamente',
() => {
    [
      { age: 0 },
      { age: 5 },
      { age: 17 },
    ].forEach(({ age }) => {
      component.setAge(age);
      expect(component.getAge()).toBe(age);
    });
  });
```

35. A continuación, se agregará un nuevo caso de prueba para personas que pueden conducir.

```
it('Casos de prueba para validar que la persona puede conducir', () => {
    [
      { age: 18 },
      { age: 21 },
      { age: 99 },
    ].forEach(({ age }) => {
      component.setAge(age);
      expect(component.canDrive()).toBe(true);
    });
  });
```

36. Repita los pasos anteriores, para agregar un caso de prueba para personas con edades que no se les permite conducir. Seguidamente, se muestra el caso de prueba y lo valores a probar son: -1, 10, 17.

```
it('Casos de prueba para validar que la persona NO puede conducir', () =>
{
    [
      { age: -1 },
      { age: 10 },
      { age: 17 },
    ].forEach(({ age }) => {
      component.setAge(age);
      expect(component.canDrive()).toBe(false);
    });
  });
```