

ANÁLISIS DE LA CLASE CALCULATOR (Capa de lógica)

En el patrón MVC, el modelo es el encargado de la **Lógica de negocio y el estado de los datos**, sin saber absolutamente nada de botones o colores. El modelo gestiona cómo se transforman los números y cómo se aplican las reglas matemáticas de una calculadora estándar.

1. Estado Interno (Propiedades Privadas)

La clase utiliza variables *private* para mantener un control estricto de los datos:

- **numAnterior y num2**: Almacenan los valores numéricos de la operación.
- **numActual**: Se maneja como string para facilitar la concatenación de dígitos (ej. escribir "1" y luego "2" para formar "12").
- **operador**: Guarda el símbolo de la operación elegida usando el tipo personalizado operador.
- **Flags (resultado, esPorcentaje)**: Variables booleanas que ayudan a la calculadora a saber en qué "estado" se encuentra (si acaba de dar un resultado o si está calculando un porcentaje).

2. Getters: Comunicación con el Exterior

Los get permiten que el **Controlador** lea datos sin poder modificarlos directamente:

- **operacionActual**: Es un método inteligente. Construye una cadena de texto que representa lo que el usuario ve en el historial (ej: "5 + 10%"). Maneja diferentes casos según si el segundo número o el porcentaje ya han sido ingresados.

3. Métodos de Entrada de Datos

ingresarNumero(num)

Es el motor de escritura.

- **Reset automático**: Si el usuario presiona un número justo después de un resultado (this.resultado === true), la calculadora limpia todo automáticamente para empezar una cuenta nueva.
- **Lógica de concatenación**: Diferencia si el usuario está escribiendo el primer número o el segundo (después de haber pulsado un operador).

ingresarOperador(op)

- **Encadenamiento**: Si el usuario ya tiene una operación pendiente (ej: "5 + 5") y presiona otro operador (ej: "x"), el modelo ejecuta primero el cálculo anterior y prepara el resultado como base para la nueva operación.
- **Números negativos**: Permite que el operador - funcione como signo si la pantalla está vacía.

ingresarDecimal()

Controla la integridad del número. Evita que el usuario ingrese múltiples puntos (ej: "5.5.5") y gestiona el caso donde el usuario empieza un número con el punto (convirtiéndolo automáticamente en "0.").

4. Lógica Matemática

```
calcular()
```

Este es el corazón de la lógica.

- Utiliza un bloque switch para decidir qué operación aritmética realizar ('+', '-', 'x', '\div').
- **Precisión:** Utiliza .toFixed(3) y vuelve a convertir a Number para evitar los errores comunes de coma flotante en JavaScript (como cuando \$0.1 + 0.2\$ da \$0.3000000000000004\$).

```
porcentaje()
```

Implementa una lógica avanzada de porcentajes:

- En **sumas y restas**: Calcula el porcentaje basado en el primer número (ej: \$100 + 10% = 110\$).
- En **multiplicación**: Simplemente divide el segundo número por 100.

5. Métodos de Limpieza

- **reset()**: Devuelve la calculadora a su estado inicial (limpieza total).
- **eliminarUltimo()**: Actúa como la tecla "Backspace", eliminando el último carácter del string actual y actualizando el valor numérico correspondiente.

EXPLICACIÓN DEL CÓDIGO

1. Tipos y constantes:

```
1 type operador = '+' | '-' | 'x' | '\div';
```

Define un **tipo unión** que limita los operadores permitidos.

Esto evita errores como usar "*" o "/" por accidente.

```
2 const MAX_DIGITOS = 9;
```

Límite máximo de dígitos que puede tener un número ingresado (simula una calculadora real).

2. Estructura general de la clase:

```
3 export class Calculator {
```

Esta clase encapsula **todo el estado y la lógica** de la calculadora.

Está pensada para ser usada desde una UI (Angular, React, etc.).

3. Atributos privados (estado interno):

Números

```
private numAnterior: number | null = null;
```

- Primer operando
- null significa que aún no se ha ingresado

```
private numActual: string = '0';
```

- Número que se muestra en pantalla
- Es string para facilitar la concatenación de dígitos

```
private num2: null | number = 0;
```

- Segundo operando

- null indica que todavía no se está ingresando

÷ Operación y estado

```
private operador: operador | null = null;
```

- Operador seleccionado (+, -, x, ÷)

```
private resultado: boolean = false;
```

- Indica si se acaba de presionar =

- Sirve para saber cuándo reiniciar la entrada

Porcentaje

```
private esPorcentaje: boolean = false;
```

- Indica si la operación actual usa porcentaje

```
private encontrarPorcentaje: number | null = null;
```

- Guarda el número original antes de convertirlo en porcentaje
- Se usa solo para mostrarlo correctamente (20%)

4. Getters (lectura segura del estado)

Número actual

```
13  get numeroActual(): string {
14    | return this.numActual;
15  }
```

Permite a la UI(Interfaz de Usuario) obtener el valor de pantalla sin modificarlo.

Estado del resultado

```
24  get hayResultado(): boolean {
25    | return this.resultado;
26  }
```

Se usa para saber si se acaba de hacer un cálculo.

Porcentaje

```
28  ↘  get existePorcentaje() {
29    |   return this.esPorcentaje;
30  }
31
32 ↘  get esNumPorcentaje() {
33    |   return this.encontrarPorcentaje;
34  }
```

Permiten a la UI saber si hay un porcentaje involucrado.

5. Getter operacionActual (texto superior)

```
get operacionActual(): string {
```

Devuelve la operación en curso para mostrarla arriba, por ejemplo:

100 + 20%

Casos que maneja:

1. No hay primer número → "
2. Solo hay primer número → numAnterior
3. Hay operador pero no segundo número → numAnterior + operador
4. Hay porcentaje → numAnterior + operador + porcentaje%
5. Operación normal → numAnterior + operador + num2

6. ingresarOperador(op)

ingresarOperador(op: operador)

Se ejecuta cuando el usuario presiona +, -, x o ÷.

Lógica clave:

1. Encadenamiento de operaciones

```
// Si ya tenemos los componentes para una operacion, calcula
if (this.numAnterior !== null && this.num2 !== null) {
    this.calcular();
}
```

Permite:

2 + 3 + 4 =

2. Número negativo inicial

```
if (this.numAnterior === null && this.numeroActual === '0' && op === '-') {
    this.numActual = '-';
    return;
}
```

3. Guarda el número actual como numAnterior

4. Resetea num2

5. Limpia el estado de porcentaje si existía

7. ingresarNumero(num)

Se ejecuta cuando se presiona un dígito (0–9).

Caso 1: venimos de =

Se reinicia todo.

Caso 2: estamos ingresando el segundo número

- Reemplaza el 0 inicial
- Concatena dígitos
- Actualiza num2

Caso 3: primer número

- Evita superar MAX_DIGITOS
- Maneja correctamente el 0 inicial

8. ingresarDecimal()

Maneja el botón .

Casos:

1. Despues de un resultado → reinicia
2. Si ya hay un punto → no hace nada

3. Si es - → convierte a 0.
4. Si es 0 → 0.
5. Caso general → agrega .

9. reset() reset()

Reinicia completamente la calculadora:

- Números
- Operador
- Resultado
- Porcentaje

Equivalente al botón **C**.

10. eliminarUltimo()

Simula el botón 

- Si hay resultado → reset
- Si queda un solo carácter → vuelve a 0
- Actualiza num2 si corresponde

11. calcular()

Ejecuta la operación cuando se presiona =.

Validaciones

Evita cálculos si:

- Falta algún operando
- El estado es inválido ('-')

Cálculo

```
switch (this.operador) {
    case '+': result = this.numAnterior + this.num2; break;
    case '-': result = this.numAnterior - this.num2; break;
    case 'x': result = this.numAnterior * this.num2; break;
    case '/': result = this.numAnterior / this.num2; break;
}
```

true

- Redondea a 3 decimales
- Guarda el resultado en pantalla
- Marca resultado =

12. Porcentaje()

Implementa el comportamiento real de una calculadora.

Ejemplos reales:

- $100 + 20\% \rightarrow 100 + (100 \times 20 / 100)$
- $100 \times 20\% \rightarrow 100 \times 0.2$

```
case '+':
case '-':
    this.num2 = this.numAnterior * this.num2 / 100;
case 'x':
```

```
this.num2 = this.num2 / 100;  
case '÷':  
    this.num2 = this.numAnterior * this.num2 / 100;
```

Resumen conceptual

- ✓ **Modelo orientado a estado**
- ✓ Simula fielmente una calculadora real
- ✓ Soporta:
 - Números negativos
 - Decimales
 - Encadenamiento de operaciones
 - Porcentajes reales
 - Borrado parcial
 - Reset total