Análisis de la Clase Controller (Capa de Control)

En el patrón MVC, esta clase actúa como el **director de orquesta** o el mediador: es quien escucha al usuario a través de la Vista y le dice al Modelo qué procesar.
El controlador es la pieza que une el **Modelo** (la lógica) y la **Vista** (la interfaz). Su objetivo es que el Modelo y la Vista no se hablen directamente entre sí, manteniendo el código organizado y fácil de mantener.

1. El Constructor: Inicialización y Suscripción

Cuando se crea la instancia del controlador:

- **Inyección de Dependencias**: Recibe las instancias de modelo y vista. Esto permite que el controlador tenga acceso a ambas herramientas.
- **Estado Inicial**: Llama a showResult para que la pantalla no aparezca vacía al cargar.
- **Activación de Escuchadores**: Ejecuta todos los métodos de "Eventos" (EventoNumeros, EventoOperador, etc.), que son los que dejan a la calculadora "atenta" a los clics del usuario.

2. Gestión de Eventos (Listeners)

EventoNumeros() y EventoOperador()

Estos métodos utilizan un ciclo forEach para recorrer los botones que la vista capturó.

- **Uso de dataset**: El controlador lee los atributos personalizados del HTML (como data-num o data-op). Esto es una excelente práctica porque separa el valor interno del botón de lo que se muestra visualmente en el texto del botón.
- **Validación de Límites**: Antes de procesar un número, consulta a la vista si el valor excede el límite. Si es así, ejecuta un **reset de seguridad**, demostrando cómo el controlador toma decisiones basadas en el estado de la aplicación.

EventoCalcular() (El botón "=")

Es uno de los métodos más importantes:

- Llama al método calcular() del modelo.
- **Sincronización de UI**: Solo si el modelo confirma que hubo un cálculo real (hayResultado), el controlador le pide a la vista que actualice el resultado y el historial de la operación, añadiendo el símbolo = para dar claridad al usuario.

eventoPorciento()

Gestiona la lógica de porcentajes. A diferencia de otros botones, este actualiza principalmente la "operación actual" para que el usuario vea el símbolo % reflejado antes de dar el resultado final.

3. El Flujo de Trabajo (Workflow)

El controlador sigue un patrón repetitivo pero vital para el MVC:

1. **Escuchar**: Detecta un clic en un botón de la Vista.

2. **Notificar**: Le dice al Modelo qué acción realizar (ej. ingresarNumero).
3. **Actualizar**: Le pide a la Vista que muestre los nuevos datos que ahora tiene el Modelo (showResult, showOperation).

4. Beneficios de este Diseño

- **Modularidad**: Si decides cambiar la lógica de cálculo (ej. añadir una función científica), solo tocas el Modelo. Si decides cambiar el diseño de la calculadora (ej. usar un tema oscuro), solo tocas la Vista. El Controlador simplemente seguirá conectando ambos.
- **Escalabilidad**: Es muy fácil añadir nuevas funciones (como un botón de raíz cuadrada) siguiendo la misma estructura de eventos.

Explicación del código:

1. Rol del Controller:

El **Controller** es el **intermediario** entre:

- **Modelo (Calculator)** → lógica y estado
- **Vista (View)** → botones, pantalla y DOM

El Controller NO calcula ni dibuja
Solo escucha eventos y coordina acciones

2. Imports

```
1   import { Calculator } from "../model/CalculatorModel.js";
2   import { View } from "../view/CalculatorView.js";
```

- Importa el **modelo** (lógica matemática)
- Importa la **vista** (DOM y UI)

Esto confirma que:

- El controlador **no conoce el HTML directamente**
- Todo pasa a través de la vista

3. Definición de la clase Controller

export class Controller {
Se exporta para poder crear una instancia desde el archivo principal (main.ts, app.ts, etc.).

3.1 Propiedades privadas

private modelo: Calculator;
private vista: View;

- modelo → instancia de Calculator
- vista → instancia de View

Son **privadas** porque solo el Controller debe manejarlas.

4. Constructor

constructor(modelo: Calculator, vista: View) {

El controlador **recibe las dependencias** (inyección de dependencias).
¿Por qué es buena práctica?

- Permite reutilizar
- Facilita pruebas
- Reduce acoplamiento

4.1 Asignación de dependencias

this.modelo = modelo;
this.vista = vista;
Se guardan las referencias internas.

4.2 Mostrar estado inicial

this.vista.showResult(this.modelo.numeroActual)

- Muestra 0 en pantalla al iniciar
- Sin tocar directamente el DOM

4.3 Registro de eventos

this.EventoOperador();
this.EventoNumeros();
this.EventoCalcular();
this.eventoDecimal();
this.eventoLimpiar();
this.eventoQuitarUltimo();
this.eventoPorciento();

Aquí el Controller se suscribe a **TODOS los eventos de la UI**.
Cada método:

- Escucha botones
- Llama al modelo
- Actualiza la vista

5. EventoOperador()
EventoOperador() {
Gestiona los botones: + - x ÷

5.1 Recorrido de botones

this.vista.btnOperadores.forEach(btn => {

- btnOperadores es un NodeList
- Se asigna un listener a cada operador

5.2 Evento click
btn.addEventListener('click', () => {
Cada click ejecuta la lógica del operador.

5.3 Validación de límite
if (this.vista.excedeLimite(this.modelo.numeroActual)) {

- Evita números demasiado largos
- La vista valida porque conoce el display

Acción si se excede:
this.modelo.reset();
this.vista.showResult('0');
this.vista.showOperation('');
return;

Resetea todo
Evita errores de estado
Sale del evento

5.4 Obtener operador desde el botón
const oper = btn.dataset.op as '+' | '-' | 'x' | '÷';

- Usa data-op del HTML
- TypeScript fuerza operadores válidos

5.5 Enviar operador al modelo

if (oper) this.modelo.ingresarOperador(oper);
El Controller **NO interpreta** el operador
Solo lo pasa al modelo

5.6 Actualizar la vista

this.vista.showResult(this.modelo.numeroActual);
this.vista.showOperation(this.modelo.operacionActual);
this.vista.habilitarNumeros();

- Muestra el número
- Muestra la operación superior
- Reactiva botones numéricos

6. EventoNumeros()
EventoNumeros() {
Maneja los botones 0–9.

6.1 Iterar botones numéricos

this.vista.btnNumeros.forEach(btn => {

6.2 Validación de límite (igual que operadores)
Evita overflow visual.

6.3 Obtener número

const numero = Number(btn.dataset.num);
Convierte el data-num a número real.
6.4 Enviar al modelo

this.modelo.ingresarNumero(numero);
El Controller **no concatena**, solo delega.

6.5 Actualizar pantalla

this.vista.showResult(this.modelo.numeroActual);
this.vista.showOperation(this.modelo.operacionActual);

7. eventoDecimal()
eventoDecimal() {
Maneja el botón .

7.1 Listener
this.vista.btnDecimal.addEventListener('click', () => {

7.2 Delegación total al modelo
this.modelo.ingresarDecimal();
El modelo valida
El controlador solo coordina

7.3 Actualización visual
this.vista.showResult(this.modelo.numeroActual);
this.vista.showOperation(this.modelo.operacionActual);

8. EventoCalcular()
EventoCalcular() {
Maneja el botón =

8.1 Click del igual
this.vista.btnIgual.addEventListener('click', () => {

8.2 Ejecutar cálculo
this.modelo.calcular();

8.3 Verificar si hubo resultado real
if (this.modelo.hayResultado) {
Evita mostrar basura si la operación era inválida.

8.4 Actualizar vista tras cálculo
this.vista.showResult(this.modelo.numeroActual);
this.vista.habilitarNumeros();
this.vista.showOperation(`${this.modelo.operacionActual} =`);

- Muestra resultado
- Permite nueva operación
- Agrega = a la operación

9. eventoLimpiar()
eventoLimpiar(){
Botón **C**

9.1 Click
this.vista.btnLimpiar.addEventListener('click',()=>{

9.2 Reset completo
this.modelo.reset();
this.vista.showResult(this.modelo.numeroActual);
this.vista.showOperation(this.modelo.operacionActual);

Resetea modelo y vista sincronizados.

10. eventoQuitarUltimo()
eventoQuitarUltimo(){
Botón ⌫

10.1 Click
this.vista.btnQuitUlt.addEventListener('click',()=>{

10.2 Delegación al modelo
this.modelo.eliminarUltimo();

10.3 Refrescar UI
this.vista.showResult(this.modelo.numeroActual);
this.vista.showOperation(this.modelo.operacionActual);

11. eventoPorciento()
eventoPorciento(){
Maneja el botón %.

11.1 Click
this.vista.btnPorciento.addEventListener('click',()=>{

11.2 Cálculo de porcentaje
this.modelo.porcentaje();
El modelo transforma el número
El Controller no hace matemáticas

11.3 Actualizar operación
this.vista.showOperation(this.modelo.operacionActual);
No se actualiza el resultado porque el cálculo final aún no se hizo.

Resumen general del Controller
Implementa **MVC puro**
No conoce el DOM directamente
No hace cálculos
No guarda estado propio
Solo:
- Escucha eventos
- Llama al modelo
- Actualiza la vista