Bulma Calculator - Code Explanation Notes

📋 HTML Structure & Bulma Classes Used

Container & Layout

```html
<div class="container">         <!-- Bulma: Centers content with responsive margins -->
<h1 class="title is-2">        <!-- Bulma: Large title styling (is-2 = 2nd largest size) -->
```

What it does:

- `container`: Bulma's responsive container class that centers content and adds appropriate margins
- `title is-2`: Bulma's typography class for headings with predefined font sizes and spacing

Calculator Structure

```html
<div class="calculator-container">    <!-- Custom wrapper with glassmorphism styling -->
<div class="display">               <!-- Custom display area for numbers -->
<div class="calculator-grid">       <!-- Custom CSS Grid for button layout -->
```

What it does:

- Custom classes handle the calculator's visual design
- Bulma doesn't have calculator-specific components, so we build on top of its foundation

---

🎨 CSS Styling Breakdown

Background & Container Effects

```css
```

```css
body {
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    min-height: 100vh;
}

.calculator-container {
    background: rgba(255, 255, 255, 0.95);   /* Semi-transparent white */
    backdrop-filter: blur(10px);             /* Glassmorphism effect */
    border-radius: 20px;                     /* Rounded corners */
    box-shadow: 0 15px 35px rgba(0, 0, 0, 0.1);  /* Depth shadow */
}
```

What it does:

- Creates a gradient background covering full viewport height
- Glassmorphism effect with semi-transparent background and blur
- Multiple shadows for realistic depth perception

Display Area Styling

```css
.display {
    background: linear-gradient(135deg, #2d3748 0%, #1a202c 100%);
    border-radius: 15px;
    box-shadow: inset 0 2px 10px rgba(0, 0, 0, 0.3);  /* Inner shadow */
}

.display::before {
    background: linear-gradient(90deg, #4299e1, #667eea, #764ba2, #4299e1);
    animation: shimmer 2s ease-in-out infinite;      /* Animated border */
}
```

What it does:

- Dark gradient background for LCD-like display appearance
- Inset shadow creates recessed/sunken effect
- Animated pseudo-element creates moving shimmer effect on top border

Button Grid System

```css
```

```
.calculator-grid {
    display: grid;
    grid-template-columns: repeat(4, 1fr);    /* 4 equal columns */
    gap: 15px;                                /* Space between buttons */
}

.zero-btn {
    grid-column: span 2;                      /* Zero button spans 2 columns */
    aspect-ratio: 2/1;                        /* Width:height = 2:1 */
}
```

What it does:

- CSS Grid creates responsive 4-column layout
- `gap` provides consistent spacing between all buttons
- Zero button spans two columns for traditional calculator look

---

🎯 Button Styling & Animation System

Base Button Styles

```css
.calculator-button {
    aspect-ratio: 1;                          /* Perfect square buttons */
    border-radius: 15px;
    cursor: pointer;
    transition: all 0.2s ease;                /* Smooth hover effects */
    position: relative;                       /* For pseudo-element positioning */
    overflow: hidden;                         /* Hide ripple effect overflow */
}
```

What it does:

- `aspect-ratio: 1` ensures all buttons are perfect squares
- `position: relative` allows absolute positioning of pseudo-elements
- `overflow: hidden` contains the ripple animation within button bounds

Hover & Press Effects

```css
```

```css
.calculator-button:hover {
    transform: translateY(-2px);            /* Lift button up */
    box-shadow: 0 8px 25px rgba(0, 0, 0, 0.15);  /* Increase shadow */
}

.calculator-button::before {
    content: '';
    position: absolute;
    width: 0; height: 0;                   /* Start invisible */
    background: rgba(255, 255, 255, 0.3);   /* Semi-transparent white */
    border-radius: 50%;                     /* Perfect circle */
    transform: translate(-50%, -50%);       /* Center the circle */
}

.calculator-button:active::before {
    width: 300px; height: 300px;            /* Expand on click */
}
```

What it does:

- Hover effect lifts buttons and increases shadow for depth
- Pseudo-element creates ripple effect that expands from center on click
- `translate(-50%, -50%)` centers the ripple effect regardless of expansion

Color-Coded Button Types

```css
```

```
.number-btn {
    background: linear-gradient(135deg, #f7fafc 0%, #edf2f7 100%);  /* Light gray */
}

.operator-btn {
    background: linear-gradient(135deg, #4299e1 0%, #3182ce 100%);  /* Blue */
}

.equals-btn {
    background: linear-gradient(135deg, #48bb78 0%, #38a169 100%);  /* Green */
}

.clear-btn {
    background: linear-gradient(135deg, #f56565 0%, #e53e3e 100%);  /* Red */
}
```

What it does:

- Visual hierarchy through color coding (numbers=light, operators=blue, etc.)
- Gradients add depth and modern appearance
- Each button type has distinct hover states with darker variations

---

⚙️ JavaScript Functionality

Calculator Class Structure

```javascript
class Calculator {
    constructor(previousOperandElement, currentOperandElement) {
        this.previousOperandElement = previousOperandElement;  // Top display
        this.currentOperandElement = currentOperandElement;    // Main display
        this.clear();                                          // Initialize
    }
}
```

What it does:

- Object-oriented approach for clean code organization
- Takes DOM elements as parameters for display updates

- Initializes with cleared state

Core Methods

Number Input Handling

javascript

```javascript
appendNumber(number) {
  if (number === '.' && this.currentOperand.includes('.')) return; // Prevent multiple decimals
  this.currentOperand = this.currentOperand.toString() + number.toString();
  this.updateDisplay();
}
```

What it does:

- Validates decimal input to prevent multiple decimal points
- Concatenates new digit to current number
- Updates display after each input

Operation Processing

javascript

```javascript
chooseOperation(operation) {
  if (this.currentOperand === '') return;        // Need a number first
  if (this.previousOperand !== '') {            // Chain operations
    this.compute();
  }
  this.operation = operation;
  this.previousOperand = this.currentOperand;   // Move current to previous
  this.currentOperand = '';                     // Clear for next number
}
```

What it does:

- Validates that there's a number to operate on
- Handles chained operations by auto-computing previous operation
- Prepares for next number input

Calculation Engine

javascript

```
compute() {
  let computation;
  const prev = parseFloat(this.previousOperand);
  const current = parseFloat(this.currentOperand);

  switch (this.operation) {
    case '+': computation = prev + current; break;
    case '-': computation = prev - current; break;
    case '×': computation = prev * current; break;
    case '÷':
      if (current === 0) {
        this.showError(); return;          // Handle division by zero
      }
      computation = prev / current; break;
  }
}
```

What it does:

- Converts string inputs to numbers for calculation
- Switch statement handles all four basic operations
- Special error handling for division by zero

Display Formatting

```javascript
getDisplayNumber(number) {
  const integerDigits = parseFloat(stringNumber.split('.')[0]);
  const decimalDigits = stringNumber.split('.')[1];

  integerDisplay = integerDigits.toLocaleString('en', {
    maximumFractionDigits: 0                  // Adds thousands separators
  });

  return decimalDigits != null ?
    `${integerDisplay}.${decimalDigits}` : integerDisplay;
}
```

What it does:

- Splits numbers into integer and decimal parts

- `toLocaleString()` adds comma separators for large numbers
- Preserves decimal places when present

🎮 Interactive Features

Keyboard Support

```javascript
document.addEventListener('keydown', function(event) {
    if (event.key >= '0' && event.key <= '9') {
        calculator.appendNumber(event.key);
    } else if (event.key === 'Enter' || event.key === '=') {
        calculator.compute();
    } else if (event.key === 'Escape') {
        calculator.clear();
    }
});
```

What it does:

- Maps keyboard keys to calculator functions
- Number keys input digits
- Enter/= triggers calculation
- Escape clears calculator

Error Handling & Animation

```javascript
showError() {
    this.currentOperandElement.textContent = 'Error';
    this.currentOperandElement.classList.add('error-message');    // Red text + shake
    setTimeout(() => {
        this.clear();                                        // Auto-clear after delay
        this.currentOperandElement.classList.remove('error-message');
    }, 1500);
}
```

What it does:

- Displays error message with red styling
- CSS class triggers shake animation
- Automatically clears error and resets after 1.5 seconds

📱 Responsive Design

Mobile Adaptations

```css
@media (max-width: 768px) {
    .calculator-container {
        margin: 1rem;                    /* Reduce margins on mobile */
        max-width: none;                  /* Remove width constraint */
    }

    .current-operand {
        font-size: 2rem;                 /* Smaller text for mobile */
    }

    .calculator-button {
        font-size: 1.1rem;               /* Smaller button text */
    }
}
```

What it does:

- Responsive breakpoint at 768px (tablet/mobile)
- Reduces margins and font sizes for smaller screens
- Removes width constraints to use full available space

🔧 Bulma Integration Summary

Bulma Classes Actually Used:

1. `container` - Responsive content centering
2. `title is-2` - Typography sizing and spacing

Why Limited Bulma Usage:

- Bulma excels at **layout components** (navbar, hero, columns, cards)
- Calculator is a **specialized interface** requiring custom styling
- Used Bulma for **foundational structure**, custom CSS for **unique functionality**
- Bulma's **responsive system** and **CSS reset** provide solid foundation

Custom CSS Additions:

- **CSS Grid** for button layout (Bulma uses Flexbox columns)
- **Glassmorphism effects** not available in Bulma
- **Complex animations** (shimmer, ripple, shake)
- **Gradient backgrounds** and **color schemes**
- **Interactive states** and **transitions**

This approach combines Bulma's **reliability** with **custom creativity** for a unique, functional calculator interface.