


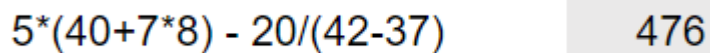
## Introduction

The task is to create a control library comprising two web controls (widgets) – NumericInput and CalcInput.

NumericInput is an input field that converts the text entered by the user to a numeric value, if possible. It could look like here:



CalcInput is similar to it, but allows the user to type arithmetical expressions like  $5*(7+8)/2-7$ , and converts them to numeric values (if possible). Besides the input field it also contains an area where it shows the calculated value. It might look like here:



You should also implement a small application that demonstrates the use of controls.

## Requirements

1. Controls are designed to work in “evergreen” browsers (Chrome, FireFox, Edge).
2. The programming language is either ES6 (ES2015) or higher, or TypeScript. If you are tricky in both languages, then give preference to TypeScript.
3. The library can be implemented as a “global” module (loaded using HTML `<script>` tag, where library members are referenced from the global namespace), or as an ESM module (loaded using “import {...} from ‘path\_to\_module’ ” like statements). It doesn’t matter and fully up to you.
4. The library may include a separate .css file that must be loaded into the application in order to use it.
5. Every control should be implemented as a **separate** ES6/TypeScript class.
6. A control is instantiated in the browser application by calling its constructor using the ‘new’ operator, and passing a DOM element (or its ID) that denotes the location of the control on a page, as a constructor parameter. For example:

HTML: `<div id="numInput" ></div>`

ES6: `let numInput = new NumericInput('numInput');`

It’s Ok to require that such an HTML element (hereinafter “host element”) must be an empty `<div>` element.

The control’s own DOM should become a child of the host element. There is no requirement to use a Shadow DOM here (Shadow DOM in terms of the DOM created by the `Element.attachShadow()` function).

7. The control library should not have a dependency on any third-party library, including JQuery.
8. If HTML `<input>` element is used as a part of the implementation, it must be the element with `type="text"`. Elements with other types, including “number” and “tel”, are prohibited.

9. The JavaScript **eval()** function cannot be used in the implementation.
10. You can create any auxiliary classes and functions, according to your needs. Give preference to solutions that produce code of smaller size, which is important for web browser applications. We strongly encourage the use of class inheritance to achieve this goal in this specific task.
11. The code should be accompanied by reasonable amount of comments written in English, which can help to simplify its understanding.

## Controls API and functionality

### Key properties

There are at least three properties in each control class:

1. **value**: number – read/write numeric property representing the evaluated result of the value entered into the control by the user. For the `CalcInput` control it's a result of the entered expression calculation.
2. **text**: string – read/write property containing the entered text.
3. **isValid**: boolean – read-only property indicating whether the text typed in the input field (the value of the 'text' property) represents a valid number/expression or not.

Both controls allow the user to enter **any** text, including invalid number or expression. On every text change the control property values should be immediately updated. The 'value' and 'isValid' property take their new value as follows:

- If the entered text represents a correct numeric (integer or float, positive or negative) value (for `NumericInput`) or a correct expression (for `CalcInput`), the 'value' property value is a number representing the entered text (the result of the expression evaluation for `CalcInput`). **isValid** is *true*.  
Expressions in `CalcInput` may include valid numbers, `+-/*` operators, and parenthesis with any depth of nesting. For example, `"-5.3"`, `"5+3*2"` or `"5*((40+7)*8) - 20/(42-37)"`
- If the text is empty, then the 'value' property value is *null*, **isValid** is *true*.
- If the text represents an invalid value/expression, then 'value' is *undefined* and **isValid** is *false*.

When the 'value' property is assigned with the new value in code, it updates the 'text' property value with the value string representation (empty string for a null value), and **isValid** becomes *true*. If 'value' is assigned with an *undefined* value, it's converted to a *null* value.

The 'text' property value is always in sync with the text in the input field.

The common rule is that any change in the property values or input field should be immediately reflected in the control state (all property values) and control's visual representation.

### Events

Each of the abovementioned properties has a corresponding "Changed" event, like `valueChanged`, `textChanged` and `isValidChanged`. Such an event is triggered when corresponding property value has changed.

Events should provide a possibility to add multiple subscribers. They can be implemented using the Observer pattern (also known as Observable/Subject or Observable/Observer, the basic information can be found here [https://en.wikipedia.org/wiki/Observer\\_pattern](https://en.wikipedia.org/wiki/Observer_pattern) ), or any other pattern of your choice.

#### Other class members

Other mandatory class members are

- constructor(hostElement) – hostElement should be an element reference or an element ID string.
- hostElement: HTMLElement – read-only property returning the element passed to the constructor
- destroy() – the method that allows the programmer to “kill” the control, by removing its content from the page’s DOM tree and releasing all the resources, to free memory and prevent from memory leaks.

#### Visual indicators

Controls should provide at least two types of visual indicators, to distinguish its *focused* and *invalid* (isValid=false) states. The states should be denoted by different borders, for example:

Non-focused, valid:

5*(2+3)	25
---------	----

Focused, valid:

5*(2+3)	25
---------	----

Non-focused, invalid:

5*(2+3	?
--------	---

Focused, invalid:

5*(2+3	?
--------	---

In case of the invalid value, the CalInput control should show some special indicator in its “value area”, for example a quotation mark symbol:

123abc	?
--------	---

#### Styling

Controls should allow the developer to change their visual style using global css rules, which can include control specific css classes in their selectors. For example, the developer should be able to change CalInput control’s look like shown below, using solely application level css rules:

$5 * (2 + 3)$	25
---------------	----

abc	?
-----	---

Control library may include a separate .css file with default css rules, in addition to .js file, and this is a reasonable requirement to the controls user to load this css file into the application to make controls properly functioning.

If control's height/width should be specified explicitly in the application, it should be done by applying height/width CSS attributes to the control's host element.

## Sample application

You should provide a simple application that demonstrates the use of controls. It should include:

- Every control, in its default style, with the following test harness for every control
  - Information about current control property values, for the 'value', 'text' and 'isValid' properties. It should be updated on every control property change.
  - A button with an associated input field next to it, which updates the control property with a value defined in the input field (using the code like `control.text = input.value`). You should provide two such button/input pairs, one for the control's 'value' property, and the other one for the 'text' property. For example:

CalcInput

$(3+5)*11$	88
------------	----

Value: 88  
Text:  $(3+5)*11$   
Valid: true

Set value	567.89
Set text	$5*(40+7*8) - 20/(42-37)$

- Version of each control with custom style, like this:

$(3+5)*11$	88
------------	----

The customization should be made solely by application level css rules. The exact styling is up to you. It should honor the requirement to provide visual hints for the focused and invalid states.

- The application should implement an Adaptive layout (a capability to adapt to different screen sizes).

It should display control "boxes" side by side on a wide screen:

Input controls test

<p>NumericInput</p> <table border="1"> <tr> <td></td> </tr> </table> <p>Value: "" Text: "" Valid: true</p> <table border="1"> <tr> <td>Set value</td> <td>123.45</td> </tr> <tr> <td>Set text</td> <td>abc</td> </tr> </table>		Set value	123.45	Set text	abc	<p>CalcInput</p> <table border="1"> <tr> <td><math>(3+5)*11</math></td> <td>88</td> </tr> </table> <p>Value: 88 Text: <math>(3+5)*11</math> Valid: true</p> <table border="1"> <tr> <td>Set value</td> <td>567.89</td> </tr> <tr> <td>Set text</td> <td><math>5*(40+7*8) - 20/(42-37)</math></td> </tr> </table>	$(3+5)*11$	88	Set value	567.89	Set text	$5*(40+7*8) - 20/(42-37)$
Set value	123.45											
Set text	abc											
$(3+5)*11$	88											
Set value	567.89											
Set text	$5*(40+7*8) - 20/(42-37)$											

Custom style

<p>NumericInput</p> <table border="1"> <tr> <td></td> </tr> </table> <p>Value: "" Text: "" Valid: true</p> <table border="1"> <tr> <td>Set value</td> <td>123.45</td> </tr> <tr> <td>Set text</td> <td>abc</td> </tr> </table>		Set value	123.45	Set text	abc	<p>CalcInput</p> <table border="1"> <tr> <td><math>(3+5)*11</math></td> <td>88</td> </tr> </table> <p>Value: 88 Text: <math>(3+5)*11</math> Valid: true</p> <table border="1"> <tr> <td>Set value</td> <td>567.89</td> </tr> <tr> <td>Set text</td> <td><math>5*(40+7*8) - 20/(42-37)</math></td> </tr> </table>	$(3+5)*11$	88	Set value	567.89	Set text	$5*(40+7*8) - 20/(42-37)$
Set value	123.45											
Set text	abc											
$(3+5)*11$	88											
Set value	567.89											
Set text	$5*(40+7*8) - 20/(42-37)$											

Destroy all

And stack them under each other on a narrow screen:  
Input controls test

Numericinput

Value:

Text: ""

Valid: true

Set value

123.45

Set text

abc

Calcinput

(3+5)\*11

88

Value: 88

Text: (3+5)\*11

Valid: true

Set value

567.89

Set text

5\*(40+7\*8) - 20/(42-37)

Custom style

Numericinput

Calcinput

(3+5)\*11

88

Destroy all

If necessary, you may use any third-party library to achieve this functionality.

4. The application should show a “Destroy all” button that, when clicked, calls the `destroy()` method for all controls on the page.
5. We should be able to run the application “in one step”, on Windows/Linux/macOS systems. For example, by opening a file in the browser directly from the local drive, or by executing a single command like “`npm run start`”, if the application setup is npm based.
6. The application should be accompanied by a readme file, in txt or md format, written in English. It should include the following information:
  - Describe the steps to run the application.
  - Any additional information regarding your implementation that you want to bring to us.