Problem 2726: Calculator with Method Chaining

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Design a

Calculator

class. The class should provide the mathematical operations of addition, subtraction, multiplication, division, and exponentiation. It should also allow consecutive operations to be performed using method chaining. The

Calculator

class constructor should accept a number which serves as the initial value of

result

.

Your

Calculator

class should have the following methods:

add

- This method adds the given number

value

to the

result

and returns the updated

Calculator

.

subtract

- This method subtracts the given number

value

from the

result

and returns the updated

Calculator

.

multiply

- This method multiplies the

result

by the given number

value

and returns the updated

Calculator

.

divide

- This method divides the

result

by the given number

value

and returns the updated

Calculator

. If the passed value is

0

, an error

"Division by zero is not allowed"

should be thrown.

power

- This method raises the

result

to the power of the given number

value

and returns the updated

Calculator

.

getResult

- This method returns the

result

.

Solutions within

10

-5

of the actual result are considered correct.

Example 1:

Input:

actions = ["Calculator", "add", "subtract", "getResult"], values = [10, 5, 7]

Output:

8

Explanation:

new Calculator(10).add(5).subtract(7).getResult() // 10 + 5 - 7 = 8

Example 2:

Input:

actions = ["Calculator", "multiply", "power", "getResult"], values = [2, 5, 2]

Output:

100

Explanation:

new Calculator(2).multiply(5).power(2).getResult() // (2 * 5) ^ 2 = 100

Example 3:

Input:

actions = ["Calculator", "divide", "getResult"], values = [20, 0]

Output:

"Division by zero is not allowed"

Explanation:

new Calculator(20).divide(0).getResult() // 20 / 0

The error should be thrown because we cannot divide by zero.

Constraints:

actions

is a valid JSON array of strings

values

is a valid JSON array of numbers

2 <= actions.length <= 2 * 10

4

1 <= values.length <= 2 * 10

4

- 1

actions[i]

is one of "Calculator", "add", "subtract", "multiply", "divide", "power", and "getResult"

First action is always "Calculator"

Last action is always "getResult"

Code Snippets

JavaScript:

```
class Calculator {

/**
* @param {number} value
*/
constructor(value) {

}

/**
* @param {number} value
* @return {Calculator}
*/
add(value){

}

/**
* @param {number} value
* @return {Calculator}
```

```
*/
subtract(value){

}

/**
* @param {number} value
* @return {Calculator}
*/
multiply(value) {

}

/**
* @param {number} value
* @return {Calculator}
*/
divide(value) {

}

/**
* @param {number} value
* @return {Calculator}
*/
power(value) {

}

/**
* @return {number}
*/
getResult() {

}
}
```

TypeScript:

```
class Calculator {

constructor(value: number) {
```

```
}

add(value: number): Calculator {

}

subtract(value: number): Calculator {

}

multiply(value: number): Calculator {

}

divide(value: number): Calculator {

}

power(value: number): Calculator {

}

getResult(): number {

}
}
```

Solutions

JavaScript Solution:

```
/**
* Problem: Calculator with Method Chaining
* Difficulty: Easy
* Tags: array, string, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

class Calculator {

/**
* @param {number} value
*/
constructor(value) {

}

/**
* @param {number} value
* @return {Calculator}
*/
add(value){

}

/**
* @param {number} value
* @return {Calculator}
*/
subtract(value){

}

/**
* @param {number} value
* @return {Calculator}
*/
multiply(value) {

}

/**
* @param {number} value
* @return {Calculator}
*/
divide(value) {

}
```

```
/**
* @param {number} value
* @return {Calculator}
*/
power(value) {

}

/**
* @return {number}
*/
getResult() {

}
}
```

TypeScript Solution:

```
/**
* Problem: Calculator with Method Chaining
* Difficulty: Easy
* Tags: array, string, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Calculator {

constructor(value: number) {

}

add(value: number): Calculator {

}

subtract(value: number): Calculator {
```

```
}

multiply(value: number): Calculator {

}

divide(value: number): Calculator {

}

power(value: number): Calculator {

}

getResult(): number {

}
}
```