You decide to test if your oddly-mathematical heating company is fulfilling its *All-Time Max, Min, Mean and Mode Temperature Guarantee™*.

Write a class `TempTracker` with these methods:

1. `insert()`—records a new temperature
2. `getMax()`—returns the highest temp we've seen so far
3. `getMin()`—returns the lowest temp we've seen so far
4. `getMean()`—returns the mean↓ of all temps we've seen so far
5. `getMode()`—returns the mode↓ of all temps we've seen so far

Optimize for space and time. **Favor speeding up the getter functions (`getMax()`, `getMin()`, `getMean()`, and `getMode()`) over speeding up the `insert()` function.**

Temperatures will all be inserted as integers. We'll record our temperatures in Fahrenheit, so we can assume they'll all be in the range `0..110`.

If there is more than one mode, return any of the modes.

Gotchas

We can get $O(1)$ time for all functions.

We can get away with only using $O(1)$ additional space in our class. If you're storing each temperature as it comes in, be careful! You might be taking up $O(n)$ space, where n is the number of temperatures we insert!

Are you trying to be fancy about returning multiple modes if there's a tie? Good idea, but *read the problem statement carefully*! Check out that last sentence!

> Failing to carefully read or listen to the problem statement is a *very* common mistake, and it *always* looks bad. Don't let it happen to you.

Breakdown

The first thing we want to optimize is our getter functions (per the instructions).

Our first thought might be to throw our temperatures into an array or linked list as they come in. With this method, getting the `maxTemp` and `minTemp` would take $O(n)$ time. It would also cost us $O(n)$ space. But we can do better.

What if we kept track of the `maxTemp` and `minTemp` *as each new number was inserted*?

That's easy enough:

JavaScript
```
function TempTracker() {
    this.minTemp = null;
    this.maxTemp = null;
}

TempTracker.prototype.insert = function(temperature) {
    if (this.maxTemp === null || temperature > this.maxTemp) {
        this.maxTemp = temperature;
    }
    if (this.minTemp === null || temperature < this.minTemp) {
        this.minTemp = temperature;
    }
};

TempTracker.prototype.getMax = function() {
    return this.maxTemp;
};

TempTracker.prototype.getMin = function() {
    return this.minTemp;
};
```

This wins us $O(1)$ time for `getMax()` and `getMin()`, while keeping $O(1)$ time for `insert()` and removing the need to store all the values.

Can we do something similar for `getMean()`?

Unlike with `minTemp` and `maxTemp`, the new temp and the previous mean won't give us enough information to calculate the new mean. What other information will we need to track?

To calculate the mean of a list of values, we need to know:

- the sum of all the values
- the total number of values

So we can augment our class to keep track of the `totalNumbers` and `totalSum`. Then we can compute the mean as values are inserted:

JavaScript
```javascript
function TempTracker() {

    // for mean
    this.totalNumbers = 0;
    this.totalSum = 0;
    this.mean = null;

    // for min and max
    this.minTemp = null;
    this.maxTemp = null;
}

TempTracker.prototype.insert = function(temperature) {

    // for mean
    this.totalNumbers++;
    this.totalSum += temperature;
    this.mean = this.totalSum / this.totalNumbers;

    // for min and max
    if (this.maxTemp === null || temperature > this.maxTemp) {
        this.maxTemp = temperature;
    }
    if (this.minTemp === null || temperature < this.minTemp) {
        this.minTemp = temperature;
    }
};

TempTracker.prototype.getMax = function() {
    return this.maxTemp;
};

TempTracker.prototype.getMin = function() {
    return this.minTemp;
};

TempTracker.prototype.getMean = function() {
    return this.meanTemp;
};
```

Can we do something similar for the mode? What other information will we need to track to compute the mode?

To calculate the mode, we need to know how many times each value has been inserted.

How can we track this? What data structures should we use?

Solution

We maintain the `maxTemp`, `minTemp`, `mean`, and `mode` as temperatures are inserted, so that each getter function simply returns an instance variable.

To maintain the `mean` at each insert, we track the `totalNumbers` and the `totalSum` of numbers inserted so far.

To maintain the `mode` at each insert, we track the total `occurrences` of each number, as well as the `maxOccurrences` we've seen so far.

JavaScript

```
function TempTracker() {

    // for mode
    this.occurrences = []; // array of 0s at indices 0..110
    for (var i = 0; i < 111; i++) {
        this.occurrences[i] = 0;
    }
    this.maxOccurrences = 0;
    this.mode = null;

    // for mean
    this.totalNumbers = 0;
    this.totalSum = 0;
    this.mean = null;

    // for min and max
    this.minTemp = null;
    this.maxTemp = null;
}
```

Type code!

```
TempTracker.prototype.insert = function(temperature) {

    // for mode
    this.occurrences[temperature]++;
    if (this.occurrences[temperature] > this.maxOccurrences) {
        this.mode = temperature;
        this.maxOccurrences = this.occurrences[temperature];
    }

    // for mean
    this.totalNumbers++;
    this.totalSum += temperature;
    this.mean = this.totalSum / this.totalNumbers;

    // for min and max
    if (this.maxTemp === null || temperature > this.maxTemp) {
        this.maxTemp = temperature;
    }
    if (this.minTemp === null || temperature < this.minTemp) {
        this.minTemp = temperature;
    }
};

TempTracker.prototype.getMax = function() {
    return this.maxTemp;
};

TempTracker.prototype.getMin = function() {
    return this.minTemp;
};

TempTracker.prototype.getMean = function() {
    return this.mean;
};

TempTracker.prototype.getMode = function() {
    return this.mode;
};
```

We don't really *need* the getter functions since they all return attributes. We could directly access the attributes!

```javascript
// function
tempTracker.getMean();

// attribute
tempTracker.mean;
```

We'll leave the getter functions in our solution because the question specifically asked for them.

But otherwise, we probably *would* use attributes instead of functions. In JavaScript we usually don't make getters if we don't *have* to, to avoid unnecessary layers of abstraction. But in Java we *would* use getters because they give us flexibility—if we need to change our logic *inside* our class, it won't change how other people *interact* with our class. Different languages, different conventions.

Complexity

$O(1)$ time for each function, and $O(1)$ space related to input! (Our `occurrences` array's size is bounded by our range of possible temps, in this case 0-110)

Did you get it right?

Yes, I'm expert on this   Not quite, review later

Like this problem? Pass it on!