JavaScript Design HashSet

Challenge

Design a HashSet without using any built-in hash table libraries.

Implement `MyHashSet` class:

- `void add(key)` Inserts the value `key` into the HashSet.
- `bool contains(key)` Returns whether the value `key` exists in the HashSet or not.
- `void remove(key)` Removes the value `key` in the HashSet. If `key` does not exist in the HashSet, do nothing.

Example

```
Input: [
           'MyHashSet','add','add','contains',
           'contains','add','contains','remove','contains'
       ]
       [
           [],[1],[2],[1],[3],[2],[2],[2],[2]
       ]
Output: MyHashSet myHashSet = new MyHashSet();
myHashSet.add(1);      // set = [1]
myHashSet.add(2);      // set = [1, 2]
myHashSet.contains(1); // return True
myHashSet.contains(3); // return False, (not found)
myHashSet.add(2);      // set = [1, 2]
myHashSet.contains(2); // return True
myHashSet.remove(2);   // set = [1]
myHashSet.contains(2); // return False, (already removed)
```

Constraints

- `0 <= key <= 10^6`
- At most 10^4 calls will be made to add, remove, and contains.

Solution

```
let MyHashSet = function() {};

MyHashSet.prototype.add = function(key) {
    this[key] = null;
};

MyHashSet.prototype.remove = function(key) {
    delete this[key];
};

MyHashSet.prototype.contains = function(key) {
    return this.hasOwnProperty(key);
};
```

Explanation

I've written a class called `MyHashSet` that represents a set of unique values. This class has three methods: `add`, `remove`, and `contains`.

The `add` method takes a `key` as a parameter and adds it as a property to the `MyHashSet` object. The `value` of the property is set to `null`.

The remove method takes a `key` as a parameter and deletes the property with that `key` from the `MyHashSet` object.

The contains method takes a `key` as a parameter and checks if the `MyHashSet` object has a property with that `key`. It returns `true` if the property exists, and `false` otherwise.

I used the prototype property of the `MyHashSet` class to define the methods. By doing so, all instances of the `MyHashSet` class will have access to these methods.

In the `add` method, the `key` is added as a property to the current instance of the `MyHashSet` object using the `this` keyword to refer to the current object.

In the `remove` method, the property with the specified `key` is deleted from the current instance of the `MyHashSet` object using the `delete` keyword.

In the `contains` method, the `hasOwnProperty` method is used to check if the current instance of the `MyHashSet` object has a property with the specified `key`. The method returns `true` if the property exists and `false` otherwise.

In summary, I've defined a class called `MyHashSet` with methods to add, remove, and check for the presence of a `key` in the set. This implementation allows for efficient management of a set of unique values.

Author: Trevor Morin