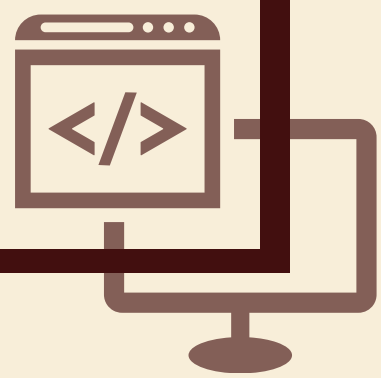
Code With Coffee

Leetcode 705-Easy

Java

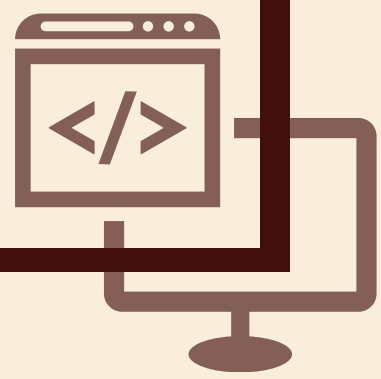
```
class MyHashSet {
    private class Node {
        int key;
        Node next;

        Node(int key) {
            this.key = key;
            this.next = null;
        }
    }

    private final Node[] set;
    private final int SIZE = 10000;

    public int getIndex(int key) {
        return key % SIZE;
    }

    public MyHashSet() {
        set = new Node[SIZE];
    }

    public void add(int key) {
        int index = getIndex(key);
        Node head = set[index];
        Node curr = head;
        while (curr != null) {
            if (curr.key == key) {
                return;
            }
            curr = curr.next;
        }
        Node newNode = new Node(key);
        newNode.next = head;
        set[index] = newNode;
    }
```



WOMEN IN CLOUD

Code With Coffee

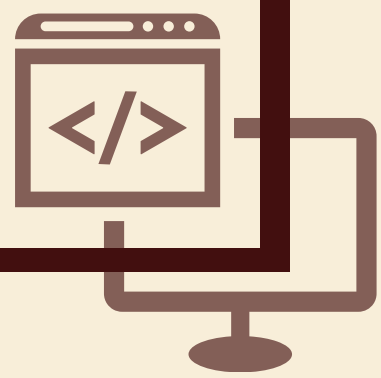
```
public void remove(int key) {
int index = getIndex(key);
Node curr = set[index];
Node prev = null;
while (curr != null) {
if (curr.key == key) {
if (prev == null) {
set[index] = curr.next;
} else {
prev.next = curr.next;
}
}
prev = curr;
curr = curr.next;
}
}
public boolean contains(int key) {
int index = getIndex(key);
Node curr = set[index];
while (curr != null) {
if (curr.key == key) {
return true;
}
curr = curr.next;
}
return false;
}
}
```



Code With Coffee

Python

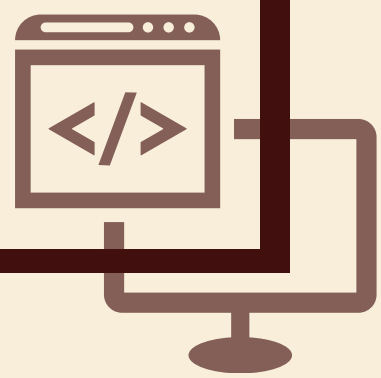
```
class MyHashSet:
    def eval_hash(self, key):
        return ((key*1031237) & (1<<20) - 1)>>5

    def __init__(self):
        self.arr = [[] for _ in range(1<<15)]

    def add(self, key: int) -> None:
        t = self.eval_hash(key)
        if key not in self.arr[t]:
            self.arr[t].append(key)

def remove(self, key: int) -> None:
    t = self.eval_hash(key)
    if key in self.arr[t]:
        self.arr[t].remove(key)

def contains(self, key: int) -> bool:
    t = self.eval_hash(key)
    return key in self.arr[t]
```



Code With Coffee

C++

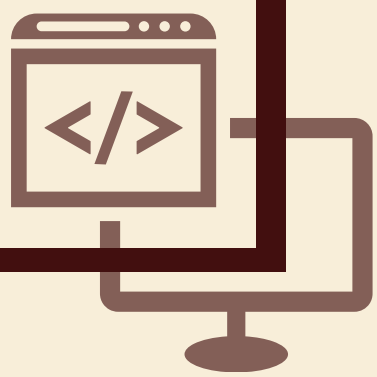
```
class MyHashSet {
private:
    std::vector<bool> mp;

public:
    MyHashSet() {
        mp.resize(1000001, false);
    }

    void add(int key) {
        mp[key] = true;
    }

    void remove(int key) {
        mp[key] = false;
    }

    bool contains(int key) {
        return mp[key];
    }
};
```



Code With Coffee

C

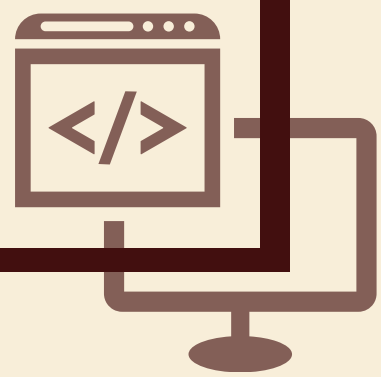
```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define SIZE 10000

// Node structure for the linked list
typedef struct Node {
    int key;
    struct Node* next;
} Node;

// HashSet structure
typedef struct {
    Node* set[SIZE];
} MyHashSet;

// Function to create a new node
Node* createNode(int key) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->key = key;
    newNode->next = NULL;
    return newNode;
}
```



Code With Coffee

C

```
// Hash function
int getIndex(int key) {
 return key % SIZE;
}

// Initialize HashSet
MyHashSet* myHashSetCreate() {
 MyHashSet* obj = (MyHashSet*)malloc(sizeof(MyHashSet));
 for (int i = 0; i < SIZE; i++) {
 obj->set[i] = NULL;
 }
 return obj;
}

// Add key
void myHashSetAdd(MyHashSet* obj, int key) {
 int index = getIndex(key);
 Node* curr = obj->set[index];
 while (curr != NULL) {
 if (curr->key == key) {
 return; // Key already exists
 }
 curr = curr->next;
 }
 Node* newNode = createNode(key);
 newNode->next = obj->set[index];
 obj->set[index] = newNode;
}
```



Code With Coffee

C

```
/ /Remove key
void myHashSetRemove(MyHashSet* obj, int key) {
 int index = getIndex(key);
 Node* curr = obj->set[index];
 Node* prev = NULL;

 while (curr != NULL) {
 if (curr->key == key) {
 if (prev == NULL) {
 obj->set[index] = curr->next;
 } else {
 prev->next = curr->next;
 }
 free(curr);
 return;
 }
 prev = curr;
 curr = curr->next;
 }
}

// Check if key exists
bool myHashSetContains(MyHashSet* obj, int key) {
 int index = getIndex(key);
 Node* curr = obj->set[index];
 while (curr != NULL) {
 if (curr->key == key) {
 return true;
 }
 curr = curr->next;
 }
 return false;
}
```