

```
package org.example;
```

```
import java.util.Objects;
```

```
public class MyHashMap<Key, Value> {
```

```
    private Key[] keys;  
    private Value[] values;  
    private boolean[] occupied;  
    private boolean[] deleted;  
    private int size;
```

```
    public MyHashMap(int capacity) {  
        keys = createArray(capacity);  
        values = createArray(capacity);  
        occupied = new boolean[capacity];  
        deleted = new boolean[capacity];  
    }
```

```
    public Value get(Key key) {  
        int index = findIndex(key);  
  
        return index == -1 ? null : values[index];  
    }
```

```
    public Value put(Key key, Value value) {  
        if (size == keys.length) {  
            resize();  
        }
```

```
        int index = findIndex(key);
```

```
        if (index != -1) {  
            Value oldValue = values[index];  
            values[index] = value;  
            return oldValue;  
        }
```

```
    }
```

```
    private int findEmptyIndex(Key key) {
```

```
        for (int index = 0; index < keys.length; index++) {  
            if (keys[index] == null ||  
                (keys[index] != null && Objects.equals(keys[index], key))) {  
                keys[index] = key;  
                values[index] = value;  
                occupied[index] = true;  
                deleted[index] = false;  
                size++;  
                return index;  
            }  
        }
```

```
    public Value remove(Key key) {  
        int index = findIndex(key);  
  
        if (index == -1) {  
            return null;  
        }
```

```
        Value oldValue = values[index];  
        occupied[index] = false;  
        deleted[index] = true;  
        size--;  
  
        return oldValue;  
    }
```

```
    private int findIndex(Key key) {  
        int index = index(key);  
        int start = index;
```

```
        while (occupied[index] || deleted[index]) {  
            if (occupied[index] && Objects.equals(keys[index], key)) {  
                return index;  
            }  
            index = (index + 1) % keys.length;  
        }
```

```
        index = (index + 1) % keys.length;  
        if (index == start) {  
            return -1;  
        }  
        return -1;  
    }
```

```
    private int findEmptyIndex(Key key) {  
        int index = index(key);  
        int start = index;  
        int deletedIndex = -1;  
  
        while (occupied[index]) {  
            if (Objects.equals(keys[index], key)) {  
                return index;  
            }  
            index = (index + 1) % keys.length;  
            if (index == start) {  
                return deletedIndex;  
            }  
        }
```

```
        return deletedIndex == -1 ? index :  
        deletedIndex;
```

```
    private int index(Key key) {  
        return Math.floorMod(key == null ? 0 : key.  
            hashCode(), keys.length);  
    }
```

```
    private void resize() {
```

```
}

private void resize() {
    Key[] oldKeys = keys;
    Value[] oldValues = values;
    boolean[] oldOccupied = occupied;

    keys = createArray(capacity: oldKeys.length * 2);
    values = createArray(capacity: oldValues.length * 2);
    occupied = new boolean[keys.length];
    deleted = new boolean[keys.length];
    size = 0;

    for (int i = 0; i < oldKeys.length; i++) {
        if (oldOccupied[i]) {
            put(key: oldKeys[i], value: oldValues[i]);
        }
    }
}

@SuppressWarnings("unchecked")
private <T> T[] createArray(int capacity) {
    return (T[]) new Object[capacity];
}
}
```