# Design Phone Directory

Design a Phone Directory which supports the following operations:

1. `get`: Provide a number which is not assigned to anyone.
2. `check`: Check if a number is available or not.
3. `release`: Recycle or release a number.

**Example:**

```
// Init a phone directory containing a total of 3 numbers: 0, 1, and 2.
PhoneDirectory directory = new PhoneDirectory(3);

// It can return any available phone number. Here we assume it returns 0.
directory.get();

// Assume it returns 1.
directory.get();

// The number 2 is available, so return true.
directory.check(2);

// It returns 2, the only number that is left.
directory.get();

// The number 2 is no longer available, so return false.
directory.check(2);

// Release number 2 back to the pool.
directory.release(2);

// Number 2 is available again, return true.
directory.check(2);
```

## Solution 1

```
    Set<Integer> used = new HashSet<Integer>();
    Queue<Integer> available = new LinkedList<Integer>();
    int max;
    public PhoneDirectory(int maxNumbers) {
        max = maxNumbers;
        for (int i = 0; i < maxNumbers; i++) {
            available.offer(i);
        }
    }

    public int get() {
        Integer ret = available.poll();
        if (ret == null) {
            return -1;
        }
        used.add(ret);
        return ret;
    }

    public boolean check(int number) {
        if (number >= max || number < 0) {
            return false;
        }
        return !used.contains(number);
    }

    public void release(int number) {
        if (used.remove(number)) {
            available.offer(number);
        }
    }
```

written by mylzsd original link here

## Solution 2

The idea is to use java's bitset and use smallestFreeIndex/max to keep track of the limit.
Also, by keeping track of the updated smallestFreeIndex all the time, the run time of get()
is spared from scanning the entire bitset every time.

```
public class PhoneDirectory {

    BitSet bitset;
    int max; // max limit allowed
    int smallestFreeIndex; // current smallest index of the free bit

    public PhoneDirectory(int maxNumbers) {
        this.bitset = new BitSet(maxNumbers);
        this.max = maxNumbers;
    }

    public int get() {
        // handle bitset fully allocated
        if(smallestFreeIndex == max) {
            return -1;
        }
        int num = smallestFreeIndex;
        bitset.set(smallestFreeIndex);
        //Only scan for the next free bit, from the previously known smallest fre
e index
        smallestFreeIndex = bitset.nextClearBit(smallestFreeIndex);
        return num;
    }

    public boolean check(int number) {
        return bitset.get(number) == false;
    }

    public void release(int number) {
        //handle release of unallocated ones
        if(bitset.get(number) == false)
            return;
        bitset.clear(number);
        if(number < smallestFreeIndex) {
            smallestFreeIndex = number;
        }
    }
}
```

written by johnyrufus16 original link here

## Solution 3

```
def __init__(self, maxNumbers):
     self.available = set(range(maxNumbers))

 def get(self):
     return self.available.pop() if self.available else -1

 def check(self, number):
     return number in self.available

 def release(self, number):
     self.available.add(number)
```

written by agave original link here