Code:

```
import java.util.*;

class Solution {
   public boolean checkSubarraySum(int[] nums, int k) {
      if (nums.length < 2) return false;

      // Special case: if k == 0, only true if two consecutive zeros exist
      for (int i = 1; i < nums.length; i++) {
         if (nums[i] == 0 && nums[i - 1] == 0) return true;
      }
      if (k == 0) return false;

      Map<Integer, Integer> map = new HashMap<>();
      map.put(0, -1); // remainder 0 seen before the array starts

      int sum = 0;
      for (int i = 0; i < nums.length; i++) {
         sum += nums[i];
         int rem = sum % k;
         if (rem < 0) rem += k; // normalize negative remainders

         if (map.containsKey(rem)) {
            if (i - map.get(rem) > 1) return true; // length ≥ 2
         } else {
            map.put(rem, i); // store the FIRST occurrence only
         }
      }
      return false;
   }
}
```

How It Works (in simple words)

We want a **continuous subarray** whose sum is divisible by k.

- Use **prefix sums**:
  sum[i] = nums[0] + nums[1] + ... + nums[i].

- If two prefix sums sum[i] and sum[j] have the **same remainder** when divided by k, then the subarray between them is divisible by k.
  Because:

(sum[i]−sum[j])%k=0
(sum[i] - sum[j]) % k = 0

- Use a **HashMap** to store the **first index** where each remainder appears.
- When we see the same remainder again, check if the distance between indices is ≥ 2.

Dry Run Example

Input:

nums = [23, 2, 4, 6, 7]
k = 6

i	nums[i]	sum	remainder = sum % k	map before	map after	Found?
				{0:-1}		
0	23	23	23 % 6 = 5	{0:-1}	{0:-1, 5:0}	No
1	2	25	25 % 6 = 1	{0:-1, 5:0}	{0:-1, 5:0, 1:1}	No
2	4	29	29 % 6 = 5	{0:-1, 5:0, 1:1}	5 already at index 0	Yes ✅
				distance = 2 - 0 = 2 (≥2) → return		

🎯 Subarray [2, 4] has sum = 6, divisible by 6.

Learning Points

1. **Why map.put(0, -1)?**
   To handle subarrays starting at index 0.
   Example: if sum[i] % k == 0, then (0..i) is valid.
2. **Why only first occurrence of remainder?**
   Because we want the longest possible subarray. If we overwrite, we lose earlier indices.
3. **Why check (i - map.get(rem) > 1)?**
   To ensure subarray length ≥ 2.
4. **Special case k == 0**
   Only valid if at least 2 consecutive zeros exist.

Final Output for Example

true