Exercise5.2 Copy Even Indices

Similar to "if" structure, nested structure also applies to loop structure. We can write "if" structure or "for" loop within any loop structure. This property gives us more flexibility to deal with different scenarios. Remember that loop is about the art of patterns. Last exercise contains only a single pattern, but more complicated problems involve more complex patterns which need us to classify more precise scenarios and find specific patterns for each precise scenario.

Problem Description:

Write a Java method int[] copyEven(int[] nums) that copy elements at even indices to a new array.

It must return the new array of the correct length with those elements inside.

Key Words:

even indices

Return a new array of the correct length

No print

Solution Design:

Again, we start with simulation. To get all values with even indices, we iterate through the array and assign each target value to a new array. Notice that the length of array can not expand once it is initialized, and the question asks to return a new array of the **correct** length. So we cannot initialize an array with a very long length, such as int[] newNums=int[10000000].

Notice the counting in plain text is different from that in one array. In an array, the first index is 0 which is an even number. Then, even indices are equivalent to the first value, the third value, the fifth value and so on. For array with even length l, our result array should store $\frac{l}{2}$ integers. For array with odd length l, our result array should sotre $\frac{l+1}{2}$ integers because the last value is also valid. Merge two cases together, the length

of result array is $\left\lfloor \frac{l+1}{2} \right\rfloor$.

Simulation cases1:

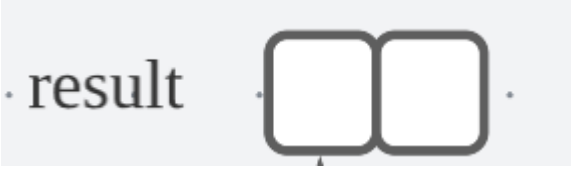
$int[]\ nums\ =\ \{1, 2, 3\};$

return: [1, 3]

Step 1:

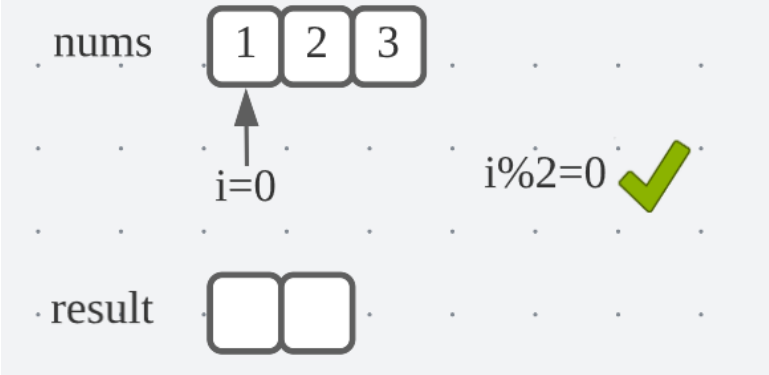
The length of $nums$ is 3 and our result array should have length 2.

$$int[]\ result = new\ int[2];$$

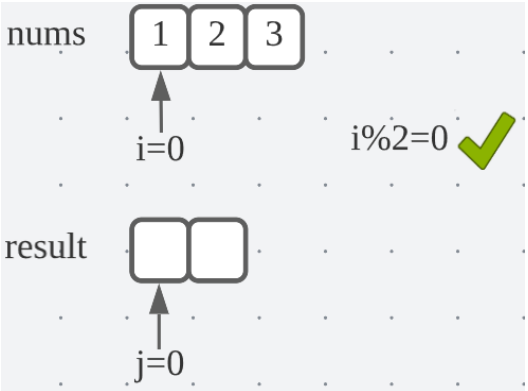


Step 2:

Iterate through $nums$, index i=0 and i%2=0, so nums[0] satisfies.

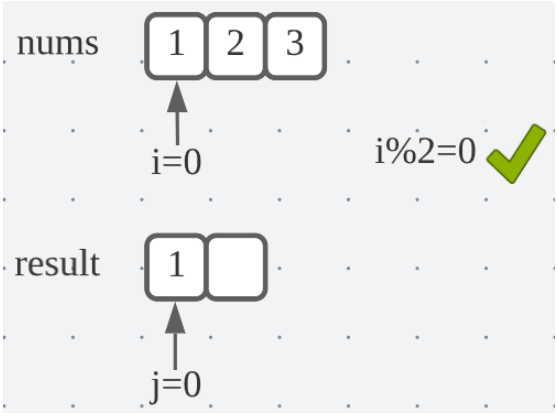


Since we also have to iterate through $result$, initialize a new integer j=0 to help us track the position in result array.



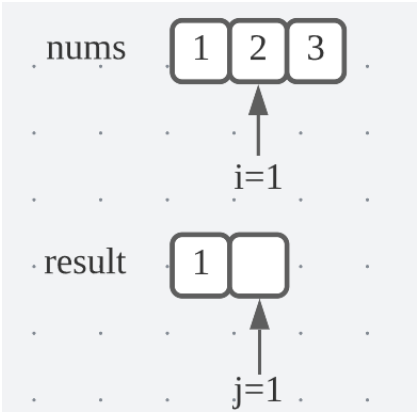
Assign $nums[0]$ to $result[j]$(equivalent to $result[0]$)

$$result[j] = nums[i]$$



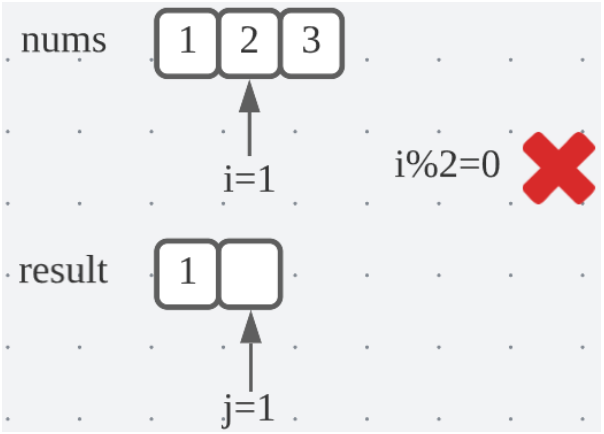
Step 3:

Increment both i and j for next loop

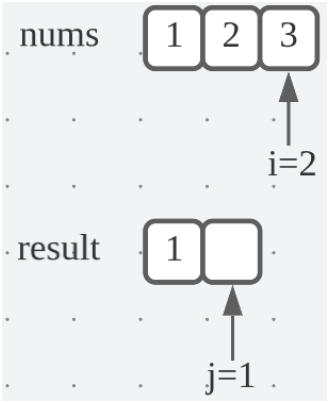


Step 4:

Now $i = 1$ and $i\%2 \neq 0$, so we don't need to assign any value to $result$.

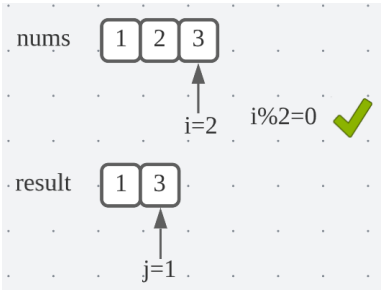


Increment only i and leave j unchanged



Step 5:

Now $i = 2$ and $i\%2 = 0$, so assign $nums[i]$ to $result[j]$.



Increment i and j

Step 6:

Index i reach the length and no more value left for us to iterate.

Done and return $result$

Simulation cases2:

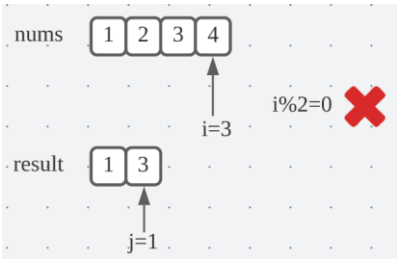
$int[]\ nums\ =\ \{1, 2, 3, 4\};$

return: [1, 3]

Step 7:

We can continue from last simulation because the length of $result$ is the same.

Increment both i and j. i=3 not satisfy. So no assigning new value and increment i



Step 7:

i=4 reaches the length of $nums$ and no more value left for us to iterate.

Done and return $results$

Pattens

Notice the exact value in array does not influence us and only indices matter.

Take simulation 2 as example

Loop	i	j	i%2=0?	Assign?
1	0	0	✓	✓
2	1	1	×	×
3	2	1	✓	✓
4	3	2	×	×

In each loop, we only assign value when i%2=0 and increment j after assignment

Thus in each loop we execute following code.

```
if (i % 2 == 0) {
    result[j] = nums[i];
    j++;
}
```

The loop structure is very clear— a classic iteration of an array

```
for (int i = 0; i < nums.length; i++) {
    if (i % 2 == 0) {
        result[j] = nums[i];
        j++;
```

```
        }
    }
```

Full implementation in files.

Optimization1

Careful observe the relationship between i and j when we assign value to $result$.

$\mathrm{i} = 2$*j ! when i satisfies, we always assign $nums[i]$ to $result[i/2]$. Thus, we can omit j and use $i/2$ to get corresponding position in result.

Optimization2

Let us think that what will program do when $i\%2 \neq 0$? Only one if judgement and nothing else! So we can skip this loop round when $i\%2 \neq 0$ even before the if judgement begins. The loop starts from i=0. Instead of incrementing it by 1, we increment it by 2. So the iteration history change from $nums[0] \rightarrow nums[1] \rightarrow nums[2]$ … to $nums[0] \rightarrow nums[2] \rightarrow nums[4]$. We only iterate value with even indices.

Optimization3

Since i=2j, why not we iterate through result.

$$result[j] = \mathrm{nums}[2*\mathrm{j}]$$

Harder version:

Please copy value with even indices inside the given array

Examples:

int[] nums = {1, 2, 3, 4};

modify nums to [1, 3, 3, 4]

int[] nums = {1, 2, 3, 4, 5 };

modify nums to [1, 3, 5, 4, 5]