Explanation on Integer.MAX_VALUE and Integer.MIN_VALUE to find min and max value in an array

Asked 9 years, 6 months ago Modified 6 years, 3 months ago Viewed 218k times

39

I don't seem to understand how `Integer.MAX_VALUE` and `Integer.MIN_VALUE` help in finding the min and max value in an array.

I understand how this method (pseudocode below) works when finding the min and max values:

```
max = A[0], min = A[0]
for each i in A
  if A[i] > max then max = A[i]
  if A[i] < min then min = A[i]
```

But as for this method, I don't understand the purpose of `Integer.MAX_VALUE` and `Integer.MIN_VALUE`:

```
import java.util.Scanner;

class MyClass {

    public static void main(String[] args) {

        int[] numbers; // declaring the data type of numbers
        numbers = new int[3]; //assigning the number of values numbers will
contain
        int smallest = Integer.MAX_VALUE, largest = Integer.MIN_VALUE;

        Scanner input = new Scanner(System.in);

        System.out.println("Please enter 3 numbers");

        for(int counter = 0; counter<numbers.length;counter++) {
            numbers[counter] = input.nextInt();
        }

        for(int i = 0; i<numbers.length; i++) {
            if(numbers[i]<smallest)
                smallest = numbers[i];
            else if(numbers[i]>largest)
                largest = numbers[i];
        }

        System.out.println("Largest is "+largest);
        System.out.println("Smallest is "+smallest);
```

```
    }
}
```

- System.out.println(Integer.MAX_VALUE) gives 2147483647
- System.out.println(Integer.MIN_VALUE) gives -2147483648

So what purpose do Integer.MIN_VALUE and Integer.MIN_VALUE serve in the comparisons?

java arrays numbers

Share
Improve this question
Follow

edited Jun 7, 2015 at 6:03

Salman Arshad
272k ● 84 ● 441 ● 533

asked Jun 6, 2015 at 17:28

Tia
1,230 ● 5 ● 22 ● 47

What do you meant "how"? They're numbers, like any other number. You compare them. – markspace Jun 6, 2015 at 17:31

`Integer.MAX_VALUE` and `Integer.MIN_VALUE` are used as a sentinel to tackle with special case – Yossarian42 Jan 9, 2021 at 8:28

3 Answers

Sorted by: Highest score (default)

44

> but as for this method, I don't understand the purpose of Integer.MAX_VALUE and Integer.MIN_VALUE.

By starting out with `smallest` set to `Integer.MAX_VALUE` and `largest` set to `Integer.MIN_VALUE`, they don't have to worry later about the special case where `smallest` and `largest` don't have a value yet. If the data I'm looking through has a `10` as the first value, then `numbers[i]<smallest` will be true (because `10` is `<` `Integer.MAX_VALUE`) and we'll update `smallest` to be `10`. Similarly, `numbers[i]>largest` will be `true` because `10` is `>` `Integer.MIN_VALUE` and we'll update `largest`. And so on.

Of course, when doing this, you must ensure that you have at least one value in the data you're looking at. Otherwise, you end up with apocryphal numbers in `smallest` and `largest`.

Note the point Onome Sotu makes in the comments:

> ...if the first item in the array is larger than the rest, then the largest item will always be Integer.MIN_VALUE because of the else-if statement.

Which is true; here's a simpler example demonstrating the problem (live copy):

```
public class Example
{
    public static void main(String[] args) throws Exception {
        int[] values = {5, 1, 2};
        int smallest = Integer.MAX_VALUE;
        int largest  = Integer.MIN_VALUE;
        for (int value : values) {
            if (value < smallest) {
                smallest = value;
            } else if (value > largest) {
                largest = value;
            }
        }
        System.out.println(smallest + ", " + largest); // 1, 2 -- WRONG
    }
}
```

To fix it, either:

1. Don't use `else`, or
2. Start with `smallest` and `largest` equal to the first element, and then loop the remaining elements, keeping the `else if`.

Here's an example of that second one (live copy):

```
public class Example
{
    public static void main(String[] args) throws Exception {
        int[] values = {5, 1, 2};
        int smallest = values[0];
        int largest  = values[0];
        for (int n = 1; n < values.length; ++n) {
            int value = values[n];
            if (value < smallest) {
                smallest = value;
            } else if (value > largest) {
                largest = value;
            }
        }
        System.out.println(smallest + ", " + largest); // 1, 5
    }
}
```

Share

Improve this answer

Follow

edited Sep 9, 2018 at 10:26

answered Jun 6, 2015 at 17:32

T.J. Crowder

1.1m ● 199 ● 2k ● 1.9k

2 @Diksha: Yes, and so the next value will be checked against them. If `5` were the second value in my example above, for instance, it would change `smallest` (because `5 < 10` is `true`) but not largest (because `5 > 10` is `false`). Then if the third value were `12`, it wouldn't change `smallest` (because `12 < 5` is `false`), but *would* change `largest` (because `12 > 10` is `true`). – T.J. Crowder Jun 6, 2015 at 21:15

1 @AravinthanK: Suppose all the numbers are negative? Starting with `max = 0` would give you the wrong result. – T.J. Crowder Dec 22, 2016 at 13:04

2 In this example, if the first item in the array is larger than the rest, then the largest item will always be Integer.MIN_VALUE because of the else-if statement. The best solution should be two if blocks testing numbers[1] against smallest and largest separately. – Onome Sotu Sep 8, 2018 at 16:23

2 @OnomeSotu - That's a **very** good point, though I think I'd argue that the best solution would be to set `smallest` and `largest` to the first value, and then loop only the remainder, keeping the `else if`. – T.J. Crowder Sep 8, 2018 at 17:14

1 @T.J.Crowder, you're correct. But if we do that, then we wouldn't have any need for the `Interger.MIN_VALUE` or `Integer.MAX_VALUE`, right? – Onome Sotu Sep 8, 2018 at 17:47

19

Instead of initializing the variables with arbitrary values (for example `int smallest = 9999, largest = 0`) it is safer to initialize the variables with the largest and smallest values representable by that number type (that is `int smallest = Integer.MAX_VALUE, largest = Integer.MIN_VALUE`).

Since your integer array cannot contain a value larger than `Integer.MAX_VALUE` and smaller than `Integer.MIN_VALUE` your code works across all edge cases.

Share

Improve this answer

Follow

edited Jun 6, 2015 at 18:32

answered Jun 6, 2015 at 17:45

Salman Arshad

272k ● 84 ● 441 ● 533

7

By initializing the min/max values to their extreme opposite, you avoid any edge cases of values in the input: Either one of min/max is in fact one of those values (in the case where the input consists of only one of those values), or the correct min/max will be found.

It should be noted that primitive types *must* have a value. If you used Objects (ie `Integer`), you could initialize value to `null` and handle that special case for the first

comparison, but that creates extra (needless) code. However, by using these values, the loop code doesn't need to worry about the edge case of the first comparison.

Another alternative is to set both initial values to the first value of the input array (never a problem - see below) and iterate from the *2nd* element onward, since this is the only correct state of min/max after one iteration. You could iterate from the 1st element too - it would make no difference, other than doing one extra (needless) iteration over the first element.

The only sane way of dealing with inout of size zero is simple: throw an `IllegalArgumentException`, because min/max is undefined in this case.

Share

Improve this answer

Follow

edited Jun 6, 2015 at 17:56

answered Jun 6, 2015 at 17:32

Bohemian ♦

424k ● 100 ● 598 ● 742

Thanks for this explanation, it makes a lot more sense to me now... – kingified Apr 17, 2021 at 5:33