Explain to me above code so I can use it in interview

The above code is a Java class called B0053, which has a single method called **maxSubArray** that takes an array of integers as input. The purpose of this method is to find the maximum sum of a subarray within the given array of integers.

The method starts by initializing two variables: **maxSum** and **currentSum**. **maxSum** is initialized to the lowest possible value for an integer, and **currentSum** is initialized to 0. These variables are used to keep track of the maximum sum of the subarray and the current sum while iterating through the array.

The method then uses a for-each loop to iterate through each element (**num**) in the array of integers. In each iteration, the method updates the **currentSum** variable by taking the maximum of the current value or the currentSum + current value. This step is crucial because it helps to keep track of the maximum sum subarray that ends at the current element.

After that, the method updates the **maxSum** variable by taking the maximum of the current **maxSum** or **currentSum**. This step is crucial because it helps to keep track of the maximum sum subarray that starts and ends at any position in the array.

Finally, the method returns the **maxSum** variable, which is the maximum sum of the subarray.

This solution has a time complexity of O(n) because we are iterating through the array once, and a space complexity of O(1) because we are using only two variables.

In an interview, you could use this code to explain how you would find the maximum sum of a subarray within an array of integers. You could also mention the time and space complexities of the solution.

Tell me in detail what does this line do

currentSum = Math.max(num, currentSum + num);

maxSum = Math.max(maxSum, currentSum);

The line **currentSum = Math.max(num, currentSum + num);** is updating the **currentSum** variable by taking the maximum of the current value **num** or the currentSum + current value **currentSum + num**.

The **Math.max(num, currentSum + num)** compares the current element of the array (**num**) with the current sum of the subarray **currentSum + num**. It then assigns the greater value to the **currentSum** variable.

The purpose of this step is to keep track of the maximum sum subarray that ends at the current element. Because the maximum sum subarray ending at the current element could either be the current element itself or the current element plus the maximum sum subarray ending at the previous element. This line of code compares these two possibilities and assigns the greater one to the currentSum variable.

Then, The line **maxSum = Math.max(maxSum, currentSum);** is updating the **maxSum** variable by taking the maximum of the current **maxSum** or **currentSum**.

The **Math.max(maxSum, currentSum)** compares the current maximum sum of the subarray **maxSum** with the current sum of the subarray **currentSum**. It then assigns the greater value to the **maxSum** variable.

The purpose of this step is to keep track of the maximum sum subarray that starts and ends at any position in the array. Because the maximum sum subarray that starts and ends at any position in the array could either be the maximum sum subarray ending at the previous position or the maximum sum subarray ending at the current position. This line of code compares these two possibilities and assigns the greater one to the maxSum variable.

So overall, these two lines of code are crucial for finding the maximum sum subarray in the given array of integers.

Tell me in detail what does this line do and why you need to do this?

int maxSum = Integer.MIN\_VALUE;

The line **int maxSum = Integer.MIN\_VALUE;** initializes the variable **maxSum** with the constant value **Integer.MIN\_VALUE**.

**Integer.MIN\_VALUE** is a predefined constant in the Java **Integer** class. It represents the smallest possible value that an **int** variable can have in Java. This value is -2147483648, and it is used as a sentinel value.

In this case, **maxSum** is being initialized with **Integer.MIN\_VALUE** because the purpose of this variable is to keep track of the maximum sum of the subarray as we iterate through the input array.

We initialize the **maxSum** with the smallest possible value for an integer because the maximum sum of the subarray could be negative and we have to take that into account.

For example, if the input array is [-2,-1,-3,-4,-1,-2,-1,-5,-4], the maximum sum of the subarray is -1, which is smaller than **Integer.MIN\_VALUE**. If we don't use **Integer.MIN\_VALUE** as the initial value for **maxSum**, it could be initialized with any positive value, that would lead to a wrong output.

So, the reason we initialize the maxSum with **Integer.MIN\_VALUE** is that it allows us to compare the current sum of the subarray with **maxSum** and assign the greater value to **maxSum** in each iteration, even if the current sum of the subarray is negative.