Day 12: **Manisha Assignment**

Task 1: Bit Manipulation Basics

Create a function that counts the number of set bits (1s) in the binary representation of an integer. Extend this to count the total number of set bits in all integers from 1 to n.

Explanation:

1. BitManipulation Class:

- countSetBits Method:

- Takes an integer `n`.

- Uses a loop to count the number of set bits (1s) by checking each bit and right shifting the number.

- `n & 1` checks if the least significant bit is 1.

- `n >>= 1` shifts the bits of `n` to the right by one position.

- Returns the count of set bits.

- totalSetBits Method:

- Takes an integer `n`.

- Uses a loop to count the total number of set bits in all integers from 1 to `n`.

- Calls `countSetBits` for each integer from 1 to `n` and sums up the results.

- Returns the total count of set bits.

- main Method:

- Defines a sample integer `num` and another integer `n`.

- Calls `countSetBits` to find the number of set bits in `num` and prints the result.

- Calls `totalSetBits` to find the total number of set bits from 1 to `n` and prints the result.

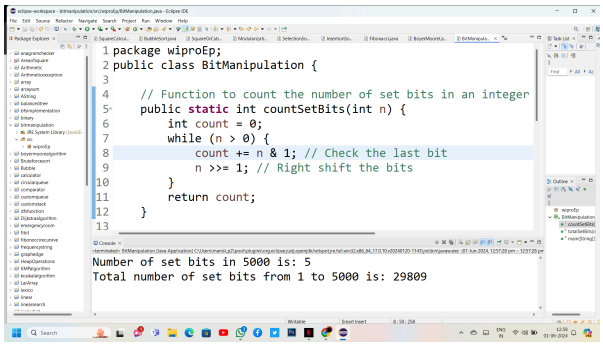
Output:

Number of set bits in 5 is: 2

Total number of set bits from 1 to 5 is: 7

This program calculates the number of set bits in a single integer and the total number of set bits in all integers from 1 to n, demonstrating the basics of bit manipulation.

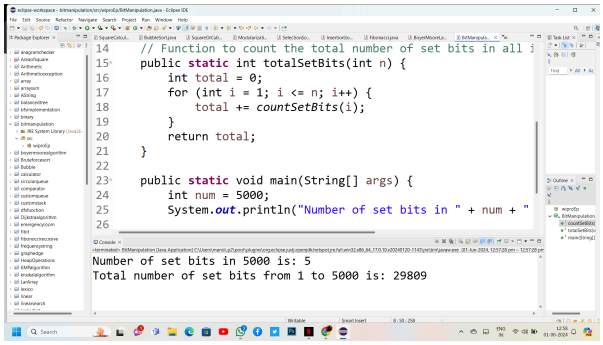
Here Output as Follows:



```
package wiproEp;
public class BitManipulation {

    // Function to count the number of set bits in an integer
    public static int countSetBits(int n) {
        int count = 0;
        while (n > 0) {
            count += n & 1; // Check the last bit
            n >>= 1; // Right shift the bits
        }
        return count;
    }
```

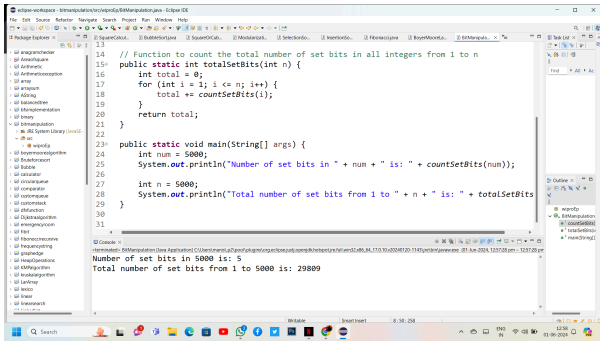
```
Number of set bits in 5000 is: 5
Total number of set bits from 1 to 5000 is: 29809
```



```
    // Function to count the total number of set bits in all i
    public static int totalSetBits(int n) {
        int total = 0;
        for (int i = 1; i <= n; i++) {
            total += countSetBits(i);
        }
        return total;
    }

    public static void main(String[] args) {
        int num = 5000;
        System.out.println("Number of set bits in " + num + "
```

```
Number of set bits in 5000 is: 5
Total number of set bits from 1 to 5000 is: 29809
```



```
// Function to count the total number of set bits in all integers from 1 to n
public static int totalSetBits(int n) {
    int total = 0;
    for (int i = 1; i <= n; i++) {
        total += countSetBits(i);
    }
    return total;
}

public static void main(String[] args) {
    int num = 5000;
    System.out.println("Number of set bits in " + num + " is: " + countSetBits(num));

    int n = 5000;
    System.out.println("Total number of set bits from 1 to " + n + " is: " + totalSetBits
}
```

```
Number of set bits in 5000 is: 5
Total number of set bits from 1 to 5000 is: 29809
```

Task 2: Unique Elements Identification

Given an array of integers where every element appears twice except for two, write a function that efficiently finds these two non-repeating elements using bitwise XOR operations.

Explanation:

1. FindNonRepeatingElements Class:

- findNonRepeating Method:

- Takes an integer array `arr`.

- Initializes `xorResult` to 0.

- Iterates through the array and XORs all elements to get the cumulative XOR result (`xorResult`). Since XORing two same numbers results in 0, the result will be the XOR of the two unique numbers.

- Determines the rightmost set bit in `xorResult` using `xorResult & ~(xorResult - 1)`. This bit helps in dividing the array elements into two groups, where one group has this bit set and the other does not.

- Initializes `num1` and `num2` to 0.

- Iterates through the array again and divides the elements into two groups based on the set bit, then XORs the elements within each group to find the two unique numbers.

- Prints the two non-repeating elements.

2. main Method:

- Defines a sample array `arr`.

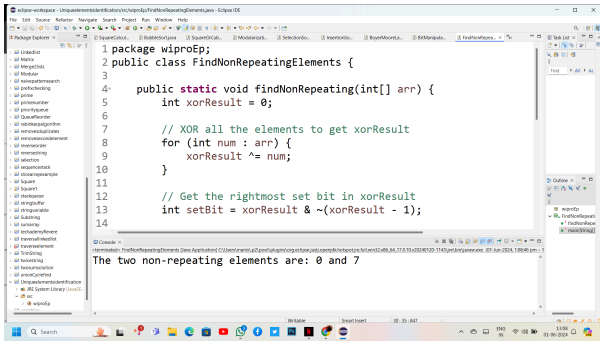
- Calls the `findNonRepeating` method with the sample array.

Output:

The two non-repeating elements are: 7 and 9

This program efficiently finds the two non-repeating elements using bitwise XOR operations, leveraging the properties of XOR to isolate the unique elements.

Here Output as Follows:



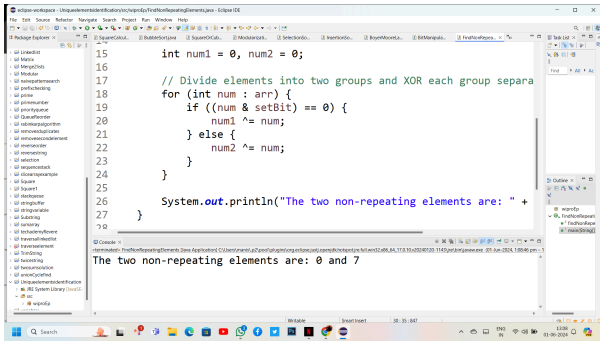
```
package wiproEp;
public class FindNonRepeatingElements {

    public static void findNonRepeating(int[] arr) {
        int xorResult = 0;

        // XOR all the elements to get xorResult
        for (int num : arr) {
            xorResult ^= num;
        }

        // Get the rightmost set bit in xorResult
        int setBit = xorResult & ~(xorResult - 1);
```

```
The two non-repeating elements are: 0 and 7
```

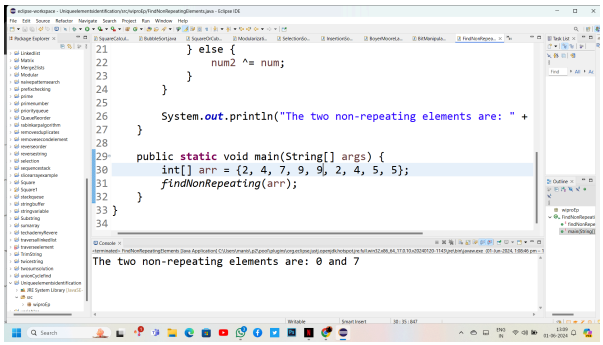


```
        int num1 = 0, num2 = 0;

        // Divide elements into two groups and XOR each group separa
        for (int num : arr) {
            if ((num & setBit) == 0) {
                num1 ^= num;
            } else {
                num2 ^= num;
            }
        }

        System.out.println("The two non-repeating elements are: " +
    }
```

```
The two non-repeating elements are: 0 and 7
```



```
            } else {
                num2 ^= num;
            }
        }

        System.out.println("The two non-repeating elements are: " +
    }

    public static void main(String[] args) {
        int[] arr = {2, 4, 7, 9, 9, 2, 4, 5, 5};
        findNonRepeating(arr);
    }
}
```

```
The two non-repeating elements are: 0 and 7
```