```
import java.util.*;

class Hamming {
	public static void main(String args[]) {
		Scanner scan = new Scanner(System.in);
		System.out.println("Enter the number of bits for the Hamming data:");
		int n = scan.nextInt();
		int a[] = new int[n];

		for(int i=0 ; i < n ; i++) {
			System.out.println("Enter bit no. " + (n-i) + ":");
			a[n-i-1] = scan.nextInt();
		}

		System.out.println("You entered:");
		for(int i=0 ; i < n ; i++) {
			System.out.print(a[n-i-1]);
		}
		System.out.println();

		int b[] = generateCode(a);

		System.out.println("Generated code is:");
		for(int i=0 ; i < b.length ; i++) {
			System.out.print(b[b.length-i-1]);
		}
		System.out.println();

		// Difference in the sizes of original and new array will give us the number of
parity bits added.
		System.out.println("Enter position of a bit to alter to check for error detection
at the receiver end (0 for no error):");
		int error = scan.nextInt();
		if(error != 0) {
			b[error-1] = (b[error-1]+1)%2;
		}
		System.out.println("Sent code is:");
		for(int i=0 ; i < b.length ; i++) {
			System.out.print(b[b.length-i-1]);
		}
		System.out.println();
		receive(b, b.length - a.length);
	}

	static int[] generateCode(int a[]) {
		// We will return the array 'b'.
```

```
        int b[];

        // We find the number of parity bits required:
        int i=0, parity_count=0 ,j=0, k=0;
        while(i < a.length) {
            // 2^(parity bits) must equal the current position
            // Current position is (number of bits traversed + number of parity bits
+ 1).
            // +1 is needed since array indices start from 0 whereas we need to
start from 1.

            if(Math.pow(2,parity_count) == i+parity_count + 1) {
                parity_count++;
            }
            else {
                i++;
            }
        }

        // Length of 'b' is length of original data (a) + number of parity bits.
        b = new int[a.length + parity_count];

        // Initialize this array with '2' to indicate an 'unset' value in parity bit locations:

        for(i=1 ; i <= b.length ; i++) {
            if(Math.pow(2, j) == i) {
            // Found a parity bit location.
            // Adjusting with (-1) to account for array indices starting from 0
instead of 1.

                b[i-1] = 2;
                j++;
            }
            else {
                b[k+j] = a[k++];
            }
        }
        for(i=0 ; i < parity_count ; i++) {
            // Setting even parity bits at parity bit locations:

            b[((int) Math.pow(2, i))-1] = getParity(b, i);
        }
        return b;
    }

    static int getParity(int b[], int power) {
```

```
        int parity = 0;
        for(int i=0 ; i < b.length ; i++) {
            if(b[i] != 2) {
                // If 'i' doesn't contain an unset value,
                // We will save that index value in k, increase it by 1,
                // Then we convert it into binary:

                int k = i+1;
                String s = Integer.toBinaryString(k);

                //Nw if the bit at the 2^(power) location of the binary value of
index is 1
                //Then we need to check the value stored at that location.
                //Checking if that value is 1 or 0, we will calculate the parity
value.

                int x = ((Integer.parseInt(s))/((int) Math.pow(10, power)))%10;
                if(x == 1) {
                    if(b[i] == 1) {
                        parity = (parity+1)%2;
                    }
                }
            }
        }
        return parity;
    }

    static void receive(int a[], int parity_count) {
        // This is the receiver code. It receives a Hamming code in array 'a'.
        // We also require the number of parity bits added to the original data.
        // Now it must detect the error and correct it, if any.

        int power;
        // We shall use the value stored in 'power' to find the correct bits to check for
parity.

        int parity[] = new int[parity_count];
        // 'parity' array will store the values of the parity checks.

        String syndrome = new String();
        // 'syndrome' string will be used to store the integer value of error location.

        for(power=0 ; power < parity_count ; power++) {
        // We need to check the parities, the same no of times as the no of parity bits
added.
```

```
			for(int i=0 ; i < a.length ; i++) {
				// Extracting the bit from 2^(power):

				int k = i+1;
				String s = Integer.toBinaryString(k);
				int bit = ((Integer.parseInt(s))/((int) Math.pow(10, power)))%10;
				if(bit == 1) {
					if(a[i] == 1) {
						parity[power] = (parity[power]+1)%2;
					}
				}
			}
			syndrome = parity[power] + syndrome;
		}
		// This gives us the parity check equation values.
		// Using these values, we will now check if there is a single bit error and then
correct it.

		int error_location = Integer.parseInt(syndrome, 2);
		if(error_location != 0) {
			System.out.println("Error is at location " + error_location + ".");
			a[error_location-1] = (a[error_location-1]+1)%2;
			System.out.println("Corrected code is:");
			for(int i=0 ; i < a.length ; i++) {
				System.out.print(a[a.length-i-1]);
			}
			System.out.println();
		}
		else {
			System.out.println("There is no error in the received data.");
		}

		// Finally, we shall extract the original data from the received (and corrected)
code:
		System.out.println("Original data sent was:");
		power = parity_count-1;
		for(int i=a.length ; i > 0 ; i--) {
			if(Math.pow(2, power) != i) {
				System.out.print(a[i-1]);
			}
			else {
				power--;
			}
		}
		System.out.println();
	}}
```

Output
Output:
Enter the number of bits for the Hamming data:
7
Enter bit no. 7:
1
Enter bit no. 6:
0
Enter bit no. 5:
1
Enter bit no. 4:
0
Enter bit no. 3:
1
Enter bit no. 2:
0
Enter bit no. 1:
1
You entered:
1010101
Generated code is:
10100101111
Enter position of a bit to alter to check for error detection at the receiver end (0 for no error):
5
Sent code is:
10100111111
Error is at location 5.
Corrected code is:
10100101111
Original data sent was:
1010101