9. IMPLEMENTING ERROR DETECTING CODE USING PARITY CHECK

#FILE NAME

Parity.java

#CODE

```
import java.util.*;

class Parity {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);

      System.out.println("Enter the size:");
      int size = in.nextInt();

      System.out.println("Enter the message as bits:");
      String mess = in.next();

      if (mess.length() != size) {
         System.out.println("Error: Message length does not match the specified size.");
         return;
      }

      int[] arr = new int[size + 1];

      for (int i = 0; i < size; i++) {
         char c = mess.charAt(i);
         if (c != '0' && c != '1') {
            System.out.println("Error: Message must contain only binary digits (0 or 1).");
            return;
         }
         arr[i] = c - '0';
      }
```

```
        int count = 0;
        for (int i = 0; i < size; i++) {
            if (arr[i] == 1) {
                count++;
            }
        }

        arr[size] = (count % 2 == 0) ? 1 : 0;

        System.out.println("The modified bits after adding parity are:");
        for (int i = 0; i < size + 1; i++) {
            System.out.print(arr[i]);
        }

        System.out.println();

        in.close();
    }
}
```

#INPUT

Enter the size:

5

Enter the message as bits:

10101

#OUTPUT

101011

10. IMPLEMENTING ERROR DETECTING CODE USING CHECKSUM

#FILE NAME

Checksum.java

#CODE

```
import java.util.Scanner;


public class Checksum {
   static String complement(String sum, int m) {
      char[] bits = sum.toCharArray();
      for (int i = 0; i < m; i++) {
         bits[i] = (bits[i] == '1') ? '0' : '1';
      }
      return new String(bits);
   }


   static String calChecksum(String[] data, int k, int m) {
      int a = Integer.parseInt(data[0], 2);
      int b = 0, c = 0;


      for (int i = 1; i < k; i++) {
         b = Integer.parseInt(data[i], 2);
         c = a + b;
         String temp = Integer.toBinaryString(c);


         if (temp.length() > m) {
            temp = temp.substring(1);
```

```
            c = Integer.parseInt(temp, 2);
            c = c + 1;
        }
        a = c;
    }

    String sum = Integer.toBinaryString(c);
    while (sum.length() < m) {
        sum = "0" + sum;
    }

    return sum;
}

static boolean validateChecksum(String[] data, int k, int m, String senderChecksum) {
    String sum = calChecksum(data, k, m);
    int s = Integer.parseInt(sum, 2);
    int sc = Integer.parseInt(senderChecksum, 2);
    s = s + sc;
    String result = Integer.toBinaryString(s);

    if (result.length() > m) {
        result = result.substring(1);
        s = Integer.parseInt(result, 2);
        s = s + 1;
    }

    String finalSum = complement(Integer.toBinaryString(s), m);
    while (finalSum.length() < m) {
```

```
        finalSum = "0" + finalSum;
    }

    System.out.println("Receiver side sum: " + Integer.toBinaryString(s));
    System.out.println("Receiver side complement: " + finalSum);
    return finalSum.equals("0".repeat(m));
}

public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    System.out.println("How many segments of data?");
    int k = input.nextInt();

    System.out.println("How many bits per segment?");
    int m = input.nextInt();

    String[] data = new String[k];
    for (int i = 0; i < k; i++) {
        System.out.println("Enter data segment " + (i + 1) + ":");
        data[i] = input.next();
    }

    String senderChecksum = complement(calChecksum(data, k, m), m);

    System.out.println("Sender side checksum value: " + senderChecksum);
    System.out.println("Receiver side complement: " + complement(senderChecksum, m));

    boolean isValid = validateChecksum(data, k, m, senderChecksum);
```

```
        System.out.println("Conclusion: " + (isValid ? "Accept Data" : "Reject Data"));

        input.close();
    }
}
```

#INPUT

How many segments of data?

4

How many bits per segment?

8

Enter data segment 1:

11010101

Enter data segment 2:

10101010

Enter data segment 3:

11110000

Enter data segment 4:

00001111

#OUTPUT

```
Sender side checksum value: 01111111
Receiver side complement: 10000000
Receiver side sum: 11111111
Receiver side complement: 00000000
Conclusion: Accept Data
```