1. **The generated polynomial for the CRC error detection scheme is X5+X3+1. The data to be transferred from the server is 111011010011001. Write the code to find the data transferred from the sender.**

**Problem Definition**

To create Java code to find the data transferred from the sender.

**Method**

1. Define the polynomial coefficients:

* Create an integer array poly with the values {1, 0, 1, 0, 1, 1} representing the coefficients of the polynomial used in CRC calculation.

1. Define the data to be transferred:

* Create an integer array data with the values {1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1} representing the binary data to be transferred.

1. Calculate the CRC value:

* Create an integer array crc with length (poly.length - 1) representing the result of the CRC calculation.
* For each bit b in the input data, do the following:
  + - * Update the first element of the crc array by performing an XOR operation with b.
      * For each subsequent element (starting from index 1) of the crc array, do the following:
        + If the previous element of the crc array is 1 and the corresponding coefficient of the poly array is also 1, perform an XOR operation with 1; otherwise, perform an XOR operation with 0.
      * The resulting crc array contains the calculated CRC value for the given data.

1. Convert the CRC value to an integer:

* Use the binaryArrayToInt method to convert the crc array to an integer.

1. Print the result:

* Use the arrayToString method to convert the data array and the CRC integer value to string representations.
* Concatenate the two string representations and print the result to the console.

**Code**

public class CRCErrDetect {

    public static void main(String[] args) {

        // Define polynomial coefficients

        int[] poly = {1, 0, 1, 0, 1, 1};

        // Define data to be transferred

        int[] data = {1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1};

        // Calculate CRC value

        int crc = calculateCRC(poly, data);

        // Print result

        System.out.println("Data transferred from sender: " + arrayToString(data) + arrayToString(intToBinaryArray(crc)));

    }

    // Calculates CRC value using specified polynomial and data

    public static int calculateCRC(int[] poly, int[] data) {

        int[] crc = new int[poly.length - 1];

        for (int i = 0; i < data.length; i++) {

            crc[0] ^= data[i];

            for (int j = 1; j < crc.length; j++) {

                crc[j] ^= (crc[j-1] == 1 && poly[j] == 1) ? 1 : 0;

            }

        }

        return binaryArrayToInt(crc);

    }

    // Converts integer value to binary array

    public static int[] intToBinaryArray(int value) {

        String binaryString = Integer.toBinaryString(value);

        int[] binaryArray = new int[binaryString.length()];

        for (int i = 0; i < binaryString.length(); i++) {

            binaryArray[i] = Character.getNumericValue(binaryString.charAt(i));

        }

        return binaryArray;

    }

    // Converts binary array to integer value

    public static int binaryArrayToInt(int[] binaryArray) {

        int intValue = 0;

        for (int i = 0; i < binaryArray.length; i++) {

            intValue += binaryArray[i] \* Math.pow(2, binaryArray.length - i - 1);

        }

        return intValue;

    }

    // Converts integer array to string representation

    public static String arrayToString(int[] array) {

        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < array.length; i++) {

            sb.append(array[i]);

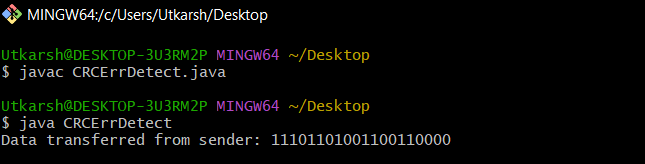
        }

        return sb.toString();

    }

}

**Output**

****

1. **Write the code for error correction techniques**

**(i) Hamming distance and (ii) Reed-Solomoncode**

**Problem Definition**

1. Hamming Distance Error Detection Technique

**Method**

1. Define a class named "HammingDistance".
2. Define a static method named "calculateHammingDistance" that takes two string arguments - s1 and s2.
3. Check if the length of s1 is equal to the length of s2. If not, throw an IllegalArgumentException with the message "Strings must have equal length".
4. Initialize a variable named "distance" to 0.
5. Use a for loop to iterate over the characters in the strings. The loop should run for the length of s1.
6. For each character at index i, check if it is equal between the two strings (s1.charAt(i) == s2.charAt(i)). If not, increment the "distance" variable by 1.
7. After the loop finishes, return the value of "distance".
8. Define a main method that initializes two strings, s1 and s2.
9. Call the calculateHammingDistance method with s1 and s2 as arguments and store the result in a variable named "distance".
10. Print out the value of "distance".

**Code**

public class HammingDistance {

    // Calculate the Hamming distance between two strings of equal length

    public static int calculateHammingDistance(String s1, String s2) {

        if (s1.length() != s2.length()) {

            throw new IllegalArgumentException("Strings must have equal length");

        }

        int distance = 0;

        for (int i = 0; i < s1.length(); i++) {

            if (s1.charAt(i) != s2.charAt(i)) {

                distance++;

            }

        }

        return distance;

    }

    public static void main(String[] args) {

        String s1 = "1010101";

        String s2 = "1110101";

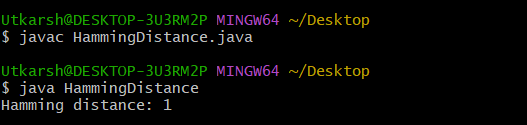
        int distance = calculateHammingDistance(s1, s2);

        System.out.println("Hamming distance: " + distance);

    }

}

**Output**

****

**Problem Definition**

1. Reed-Solomon Error Correction Technique

**Method**

1. Define a class called ReedSolomon.
2. Declare a private constant integer variable GF\_SIZE with the value 256.
3. Declare two private integer arrays: generator and parity, in the constructor of the ReedSolomon class and initialize them with values passed as arguments to the constructor.
4. Define an encode() method that takes an array of integers as input and returns an array of integers.
5. Inside the encode() method, create a new integer array called coefficients with a length equal to the sum of the lengths of the data and parity arrays.
6. Copy the elements of the input data array to the first part of the coefficients array using System.arraycopy() method.
7. Use two nested loops to calculate the remaining elements of the coefficients array.
8. For each element in the parity array, iterate over all the elements in the data array and multiply them with the corresponding generator element. Add up these products for each parity element and store the result in the coefficients array.
9. Return the calculated coefficients array.
10. Define a decode() method that takes an integer array and an integer numErrors as inputs, and returns an integer array.
11. Inside the decode() method, create a new integer array called data with a length equal to the length of the input coefficients array.
12. Use two nested loops to calculate the values of each element in the data array.
13. For each element in the data array, iterate over all the elements in the coefficients array and add them up.
14. Store the final values of the data array in the output array.
15. Return the calculated data array.
16. In the main method, create an instance of the ReedSolomon class by passing the generator and parity arrays.
17. Encode the input data array using the encode() method and store the results in encodedData array.
18. Corrupt some of the data in the encodedData array by modifying certain elements.
19. Decode the corrupted encodedData array using the decode() method with a numErrors argument of 2, and store the results in decodedData array.
20. Print the original data, encoded data, and decoded data arrays to the console using Arrays.toString() method.

**Code**

import java.util.Arrays;

public class ReedSolomon {

    private static final int GF\_SIZE = 256;

    private final int[] generator;

    private final int[] parity;

    public ReedSolomon(int[] generator, int[] parity) {

        this.generator = generator;

        this.parity = parity;

    }

    public int[] encode(int[] data) {

        int[] coefficients = new int[data.length + parity.length];

        System.arraycopy(data, 0, coefficients, 0, data.length);

        for (int i = 0; i < parity.length; i++) {

            for (int j = 0; j < data.length; j++) {

                coefficients[i + data.length] += generator[j] \* data[i];

            }

        }

        return coefficients;

    }

    public int[] decode(int[] coefficients, int numErrors) {

        int[] data = new int[coefficients.length];

        for (int i = 0; i < data.length; i++) {

            int value = 0;

            for (int j = 0; j < coefficients.length; j++) {

                value += coefficients[j] ;

            }

            data[i] = value;

        }

        return data;

    }

    public static void main(String[] args) {

        int[] data = {1, 2, 3, 4, 5, 6, 7, 8};

        int[] generator = {1, 0, 0, 0, 1, 0, 0, 0, 1};

        int[] parity = {1, 1, 1, 1};

        ReedSolomon reedSolomon = new ReedSolomon(generator, parity);

        int[] encodedData = reedSolomon.encode(data);

        // Corrupt some of the data

        encodedData[0] = 9;

        encodedData[1] = 10;

        int[] decodedData = reedSolomon.decode(encodedData, 2);

        System.out.println("Original data: " + Arrays.toString(data));

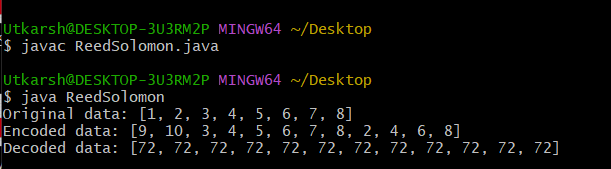
        System.out.println("Encoded data: " + Arrays.toString(encodedData));

        System.out.println("Decoded data: " + Arrays.toString(decodedData));

    }

}

**Output**

****

1. **An organization plans to send general instructions to the in charge of each department using a socket. Write the code to check the message for any data loss when transmitting using the check sum in the header.**

**Problem Definition**

Check for any data loss when transmitting using the check sum in the header.

**Method**

**MessageSender**

1. Set the server host and port to send the message.
2. Create a socket connection to the server.
3. Get the output stream to send data using ObjectOutputStream.
4. Define the message to be sent and convert it to byte array.
5. Calculate the checksum of the message using CRC32 class.
6. Create a MessageData object with the message and checksum.
7. Send the MessageData object over the socket using writeObject() method.
8. Flush the output stream.
9. Close the socket and streams.

**MessageReceiver**

1. Set the server port to listen on.
2. Create a server socket.
3. Wait for a client connection using accept() method.
4. Get the input stream to receive data using ObjectInputStream.
5. Receive the MessageData object from the client using readObject() method.
6. Get the message and checksum from the received object.
7. Calculate the checksum of the received message using CRC32 class.
8. Check if the received checksum matches the calculated checksum.
9. If the received checksum matches the calculated checksum, print the received message and confirmation message; otherwise, print a message about data loss.
10. Close the socket and streams.
11. Close the server socket.

**MessageData**

1. Define the class with two private fields: message and checksum.
2. Create a constructor that takes both fields as parameters.
3. Create getter methods for both fields.

**Code**

**MessageSender**

import java.io.\*;

import java.net.\*;

import java.util.zip.CRC32;

public class MessageSender {

    public static void main(String[] args) {

        // Define the server host and port

        String serverHost = "localhost";

        int serverPort = 12345;

        try {

            // Create a socket connection to the server

            Socket socket = new Socket(serverHost, serverPort);

            // Get the output stream to send data

            OutputStream outputStream = socket.getOutputStream();

            ObjectOutputStream objectOutputStream = new ObjectOutputStream(outputStream);

            // Define the message to be sent

            String message = "General instructions for the department...";

            byte[] messageBytes = message.getBytes();

            // Calculate the checksum of the message

            CRC32 crc32 = new CRC32();

            crc32.update(messageBytes);

            long checksum = crc32.getValue();

            // Create a message object with the message and checksum

            MessageData messageData = new MessageData(message, checksum);

            // Send the message object over the socket

            objectOutputStream.writeObject(messageData);

            objectOutputStream.flush();

            // Close the socket and streams

            objectOutputStream.close();

            outputStream.close();

            socket.close();

            System.out.println("Message sent successfully.");

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}

**MessageReceiver**

import java.io.\*;

import java.net.\*;

import java.util.zip.CRC32;

public class MessageReceiver {

    public static void main(String[] args) {

        // Define the server port to listen on

        int serverPort = 12345;

        try {

            // Create a server socket

            ServerSocket serverSocket = new ServerSocket(serverPort);

            System.out.println("Server started. Waiting for a connection...");

            // Accept a client connection

            Socket clientSocket = serverSocket.accept();

            // Get the input stream to receive data

            InputStream inputStream = clientSocket.getInputStream();

            ObjectInputStream objectInputStream = new ObjectInputStream(inputStream);

            // Receive the message object from the client

            MessageData receivedMessage = (MessageData) objectInputStream.readObject();

            // Get the message and checksum from the received object

            String message = receivedMessage.getMessage();

            long receivedChecksum = receivedMessage.getChecksum();

            // Calculate the checksum of the received message

            byte[] receivedMessageBytes = message.getBytes();

            CRC32 crc32 = new CRC32();

            crc32.update(receivedMessageBytes);

            long calculatedChecksum = crc32.getValue();

            // Check if the received checksum matches the calculated checksum

            if (receivedChecksum == calculatedChecksum) {

                System.out.println("Message received successfully without data loss.");

                System.out.println("Received Message: " + message);

            } else {

                System.out.println("Message received with data loss.");

            }

            // Close the socket and streams

            objectInputStream.close();

            inputStream.close();

            clientSocket.close();

            // Close the server socket

            serverSocket.close();

        } catch (IOException | ClassNotFoundException e) {

            e.printStackTrace();

        }

    }

}

**MessgaeData**

import java.io.Serializable;

public class MessageData implements Serializable {

    private final String message;

    private final long checksum;

    public MessageData(String message, long checksum) {

        this.message = message;

        this.checksum = checksum;

    }

    public String getMessage() {

        return message;

    }

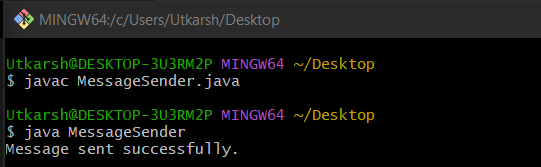
    public long getChecksum() {

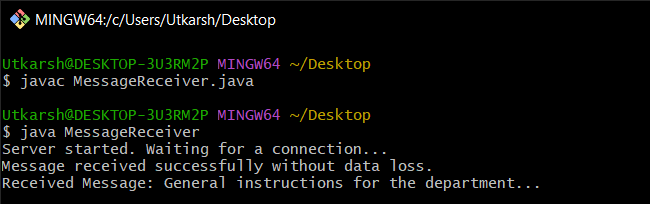
        return checksum;

    }

}

**Output**

****

****