Student Name: Asem Diab	Reg NO :12111959
Instructor Name: 1	Project Title: hammaing Code with JAVA

- **Abstract:**

This Report Speaks about Hamming Code implementation by Java. The project works as a small system of communication to send files.

- **Introduction**:

As we know, working in technology fields requires the choice between the different solutions so Hamming Code may be a solution for some problems that when you send data and if there is an error in it can't resend it or it's more expensive or needs so much time.

- **Theory**:

Hamming code is an error-correcting code that can detect and correct single-bit errors in data. It was developed by Richard Hamming in the early 1950s. The primary purpose of Hamming code is to add redundant bits to data so that errors can be detected and corrected when the data is read.

Here's a brief overview of how Hamming code works:

1. Adding Redundant Bits: Additional bits (parity bits) are added at specific positions for a block of data bits. The number of parity bits is determined by the formula 2^r >= m + r + 1, where 'm' is the number of data bits, and 'r' is the number of parity bits.

2. Calculating Parity Bits: Parity bits are calculated based on specific combinations of data bits. Each parity bit covers a specific set of data bits, and its value is set to ensure that the total number of set bits (1s) in the covered positions, including the parity bit itself, is either even (for even parity) or odd (for odd parity).

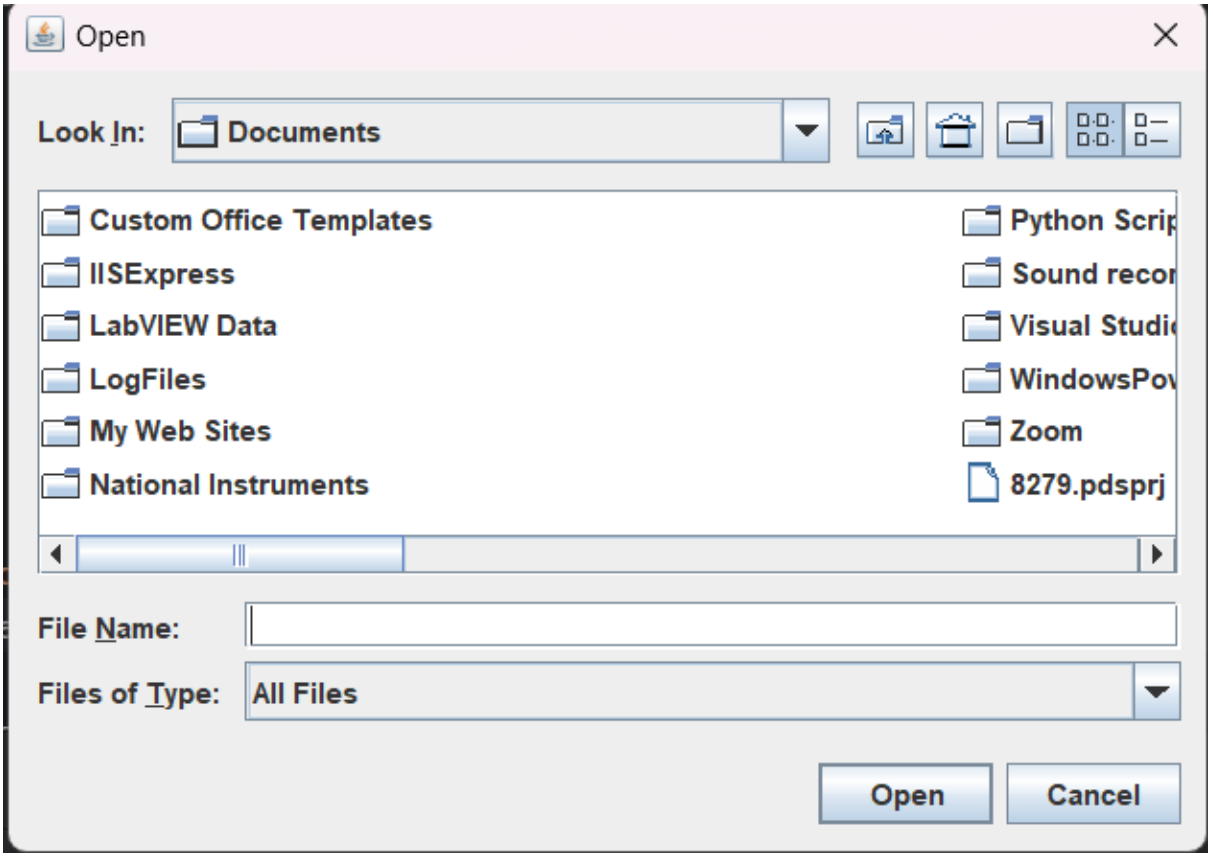
3. Error Detection and Correction: When the data is transmitted or stored, the receiver checks the parity bits. If an error is detected, the parity bits can be used to identify and correct the erroneous bit.

Hamming code is widely used in computer memory systems and communication systems where the accuracy of transmitted data is crucial. It provides a balance

between error detection and correction capabilities and the overhead of additional bits.

- **Project images:**

input window to enter file(image,video, txt,...or etc)



sample to steps:

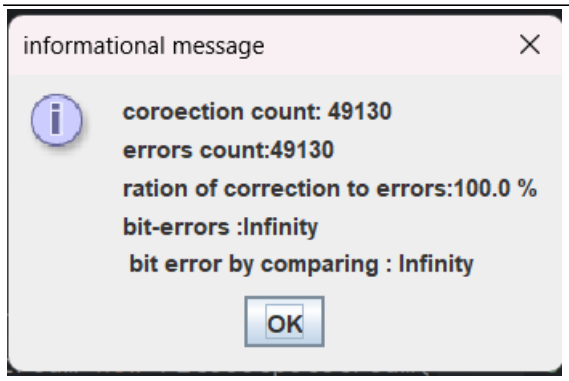
data

convert 8 bits to 12 bits

sit parity bits

Output information about Proccess

```
00000111
000000110100
000000110100
```



- **project Code**

```java
package org.example;
import javax.swing.*;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;
public class Main {

    5 usages
    static int errors=0;
    6 usages
    static  int correctingErrors=0;
    1 usage
    static int [] build_hamm(int []data){...}
    1 usage
    public static int [][] readData(String path) throws IOException{...}
    1 usage
    public static void ToByte(int number,int []arr){...}
    3 usages
    public static int  ToNumber(int []arr){...}

    2 usages
    public static int [][] produceMassage(int [][]arr){...}
```

```java
public static int correctMassage(int [][]massge){
    int x=0;
    int []errorPostion=new int[8];
    for (int i=0;i<massge.length;i++){
        errorPostion[0]=massge[i][0]^massge[i][2]^massge[i][4]^massge[i][6]^massge[i][8]^massge[i][10];
        errorPostion[1]=massge[i][1]^massge[i][2]^massge[i][5]^massge[i][6]^massge[i][9]^massge[i][10];
        errorPostion[2]=massge[i][3]^massge[i][4]^massge[i][5]^massge[i][6]^massge[i][11];
        errorPostion[3]=massge[i][7]^massge[i][8]^massge[i][9]^massge[i][10]^massge[i][11];

        int ErrorIndex=ToNumber(errorPostion);
        if(ErrorIndex>0){
            x++;
            try {


            System.out.println(ErrorIndex);
            massge[i][ErrorIndex-1]^=1;
            correctingErrors++;
            System.out.println("coooe");}
            catch (Exception exception){
                System.out.println("exception this error can't be soved ");
            }
        }
    }
    correctingErrors=x;
    System.out.println("total errors that hammming correct : "+ x);
    System.out.println("Ratio of correction to errors count"+(double)x/errors);
    return x;

}
```

```java
1 usage
public static int [][] extractData(int [][]massage){
    int [][]data=new int[massage.length][];
    correctingErrors=correctMassage(massage);
    for(int i=0;i<data.length;i++){

        data[i]=new int [8];

        int k=0;
        for(int j=2;j<12;j++){
            if(j==7||j==3){
              continue;
            }
            data[i][k]=massage[i][j];
            k++;
        }
    }
    return data;
}
```

```
public static void addNoise(int [][]massage){
    int total_errors=0;
    for(int j=0;j<massage.length;j++){

    int countOfErrors=(int)Math.floor(4*Math.random()+1);
    countOfErrors*=(int)Math.floor(100000*Math.random()+1);
    int [] noise=new int[8];
    int paratial_errors=0;
    System.out.println(countOfErrors);

        int noiseIndex=(int)Math.floor(8 * Math.random());

        if(countOfErrors==10000){

        massage[j][noiseIndex]^=1;
        massage[j][(noiseIndex+1)%8]^=1;
        total_errors+=2;

    }
    else if(countOfErrors>0){
            massage[j][noiseIndex]^=1;
            total_errors++;
    }
    }
```

```
public static void main(String[] args) throws IOException {

    JFileChooser fileChooser = new JFileChooser();

    int returnValue = fileChooser.showOpenDialog( parent: null);

    if (returnValue == JFileChooser.APPROVE_OPTION) {
        // User selected a file
        System.out.println("Selected file: " + fileChooser.getSelectedFile());
        int [][] msaa=readData(fileChooser.getSelectedFile().getPath());
                int data[][]=produceMassage(msaa);
                int cop
        addNoise(data);
        System.out.prin
        String extansion=fileChooser.getSelectedFile().getName();
        build_file(data,extansion.substring( beginIndex: extansion.lastIndexOf( ch: '.')+1));
        String str="";
        int [][] recivedDtata=extractData(data);
        str+=("coroection count: "+correctingErrors);
        str+=("\nerrors count:"+errors);
        str+=("\nration of correction to errors:"+ 100*correctingErrors/(double)errors +" % ");
        str+=("\nbit-errors :" + (data.length*12)/(double)(errors-correctingErrors) +" \n ");
        str+=("bit error by comparing : "+8.0*data.length/compare(msaa,recivedDtata));
        JOptionPane.showMessageDialog( parentComponent: null,str, title: "informational message",JOptionPane.INFORMATION_MESSAG
    } else {
        // User canceled the file selection
        System.out.println("File selection canceled.");
    }
}
```

int[][] data = produceMassage(msaa)
hammaningCode

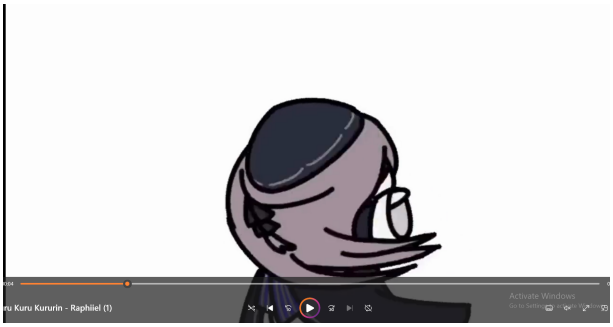
```
203 @   static int compare( int n1[][],int [][]n2){
204         int x=0;
205         double percent=0;
206
207        System.out.println("n1 ["+n1.length+" * "+n1[0].length+"\n n2 ["+n2.length+" * "+n2[0].length);
208
209         for(int i=0;i<n1.length;i++){
210             for(int j=0;j< n2[0].length;j++){
211                 if(n1[i][j]!=n2[i][j])
212                 {  x++;
213                 }}}
214        return x;}
    1 usage
215 @   private static void build_file(int[][] ints,String path) throws IOException {
216         FileOutputStream fileOutputStream=new FileOutputStream( name: "output1."+path);
217         for(int i=0;i< ints.length;i++)
218         fileOutputStream.write(ToNumber(ints[i]));
219
220         fileOutputStream.flush();
221         fileOutputStream.close();
222     }
223
224 }
```

- **Samples:**

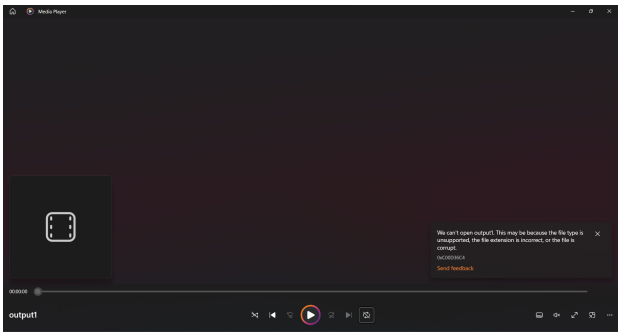
we send it with at least one error per byte so after that with Hamming Code:

1. **video**:

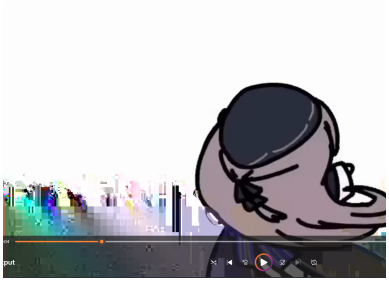
file before sending



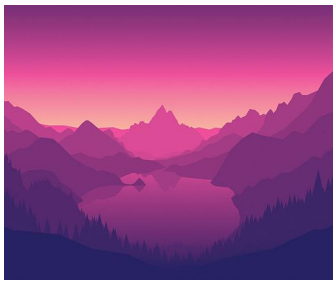
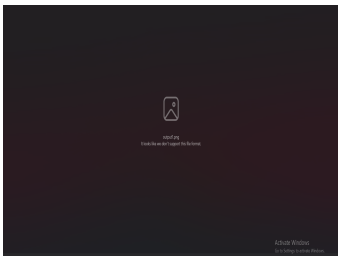
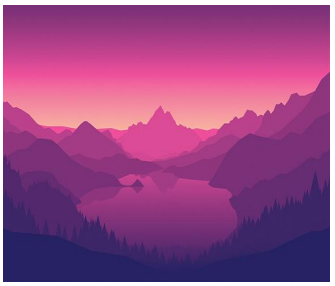
video after receiving before correction



after correction:



2. **image**:

input	**data with noise**	**data after correct**
		

3. **text:**

   we send “Asem” and after that, the data becomes BJAN due to noise. at last we corrected it and we have 50% data is correct (AjAm)

- **<u>Conclusion:</u>**

the hamming code is good for correcting single-bit errors but it can’t correct burst errors. modifying the hamming code may correct it. the many examples lead to its efficiency of it due to the most of the samples have high correcting errors. Lastly, if can

ensure that a burst error won't happen the Hamming code will be a good choice to correct data.