Design of the System

The main class start the RecordPlayback1 constructor. Here nothing happens until the program reaches captureAudio();

captureAudio();

```
private void captureAudio() {

    try {
    Mixer.Info[] mixerInfo = AudioSystem.getMixerInfo();    //get available mixers
    System.out.println("Available mixers:");
    Mixer mixer = null;
    for (int cnt = 0; cnt < mixerInfo.length; cnt++) {
        System.out.println(cnt + " " + mixerInfo[cnt].getName());
        mixer = AudioSystem.getMixer(mixerInfo[cnt]);
        Line.Info[] lineInfos = mixer.getTargetLineInfo();

        if (lineInfos.length >= 1 && lineInfos[0].getLineClass().equals(TargetDataLine.class)) {
            System.out.println(cnt + " Mic is supported!");
            break;
        }
    }

    audioFormat = getAudioFormat();     //get the audio format
    DataLine.Info dataLineInfo = new DataLine.Info(TargetDataLine.class, audioFormat);

    targetDataLine = (TargetDataLine) mixer.getLine(dataLineInfo);
    targetDataLine.open(audioFormat);
    targetDataLine.start();
    //send audio to speaker
    DataLine.Info dataLineInfo1 = new DataLine.Info(SourceDataLine.class, audioFormat);
    sourceDataLine = (SourceDataLine) AudioSystem.getLine(dataLineInfo1);
    sourceDataLine.open(audioFormat);
    sourceDataLine.start();

    //Setting the maximum volume
    FloatControl control = (FloatControl)sourceDataLine.getControl(FloatControl.Type.MASTER_GAIN);
    control.setValue(control.getMaximum());


    } catch (LineUnavailableException e) {
    System.out.println("LineUnavailableException!"+e);
    System.exit(0);
    }

}
```

The audio part of this application is handled by this. In this code, we are using captureAudio( ) function to print out the audio information. The same function contains the required codes to get audio from the input device (i.e:microphone) This keeps capturing the audio but won't send it to the receiver

Then there is a selection to be the sender or a receiver

Send

If send is selected then the there we should input the multicast IP address(to create packet) and the port number. After that comes the token checker. The request is only sent if the token

inserted “send”. Then it calls the SendRequest class along with the IP address, port number and the token.

SendRequest();

```
162 //-----------------------------------
163     public class SendRequest{
164         SendRequest(String Address,int sendPort,String token){
165 
166         // client side that send the request packets
167         int i;
168         try{
169 
170             add = InetAddress.getByName("224.2.2.0");
171             DatagramSocket socket1 = new DatagramSocket();
172             byte[] buffer = new byte[65535];
173             String mess =token;
174             buffer = mess.getBytes();
175             DatagramPacket packet = new DatagramPacket(buffer, buffer.length, add,6789);
176             socket1.send(packet);
177             socket1.close();
178 
179         }
180         catch(IOException io){System.out.println("IOException2!"+io);}
181         }
182     }
183 //-----------------------------------
```

The sending process of this program happens when client initiates the communication with the server by entering “2” as the input . The java application will instantiate a multicast socket as shown in Figure 8 . This socket will be bind with the port number of the server which is 6789. The code shows that, after the socket is created, the program will fill the buffer with some data, followed by the creation of datagram packet .

CaptureThread class;

```
227 //-------------------------------------
228
229     // sending audio to server(not our)
230     class CaptureThread extends Thread {
231
232     int count=0;
233
234         CaptureThread(String Address,int sendPort){}
235
236         byte tempBuffer[] = new byte[1024];
237
238           public void run() {
239
240             try {
241                     DatagramSocket client_socket = new  DatagramSocket();
242
243                     InetAddress IPAddress =InetAddress.getByName(Address);
244
245                     // scheduler to monitor statistics every minute
246                 final ScheduledExecutorService service = Executors.newSingleThreadScheduledExecutor();
247                 service.scheduleWithFixedDelay(new Runnable() {
248                         @Override
249                         public void run() {
250                             //if you wont to see number of packets per 10 seconds remove the comment
251                            // System.out.println("total send " + count );
252                             count = 0;
253                         }
254                 }, 5, 5, TimeUnit.SECONDS);
255
256                 while (true) {
257
258                     int cnt =targetDataLine.read(tempBuffer, 0,tempBuffer.length);//put audio to tempBuffer
259
260                     byte sendBuffer[] = new byte[2100];
261
262                     Reading1 p = new Reading1(tempBuffer,count);//creating a object to serialize
263
264                         byte[] array= Reading1.Serialize(p);//serialize the object
265
266                             sendBuffer = Arrays.copyOf(array,array.length);
267
268                   DatagramPacket send_packet = new DatagramPacket(sendBuffer, sendBuffer.length,IPAddress,sendPort);
269
270                   client_socket.send(send_packet);
271
272                   count++;
273                }
274
275
276             } catch (Exception e) {
277                 System.out.println("Exception1!"+e);
278                 System.exit(0);
279             }
280         }
281     }
282
283 //-------------------------------------
```

In order to send audio to the server, we have used the following code presented below. We are using Reading1 class to serialize the object.

The mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object. Our object packets includes sequence number and tempBuffer

The audio will be sent using a datagram with it’s respective IP address and port number .

Reading1 class;

```
14    public class Reading1 implements Serializable {
15
16     public final byte []tempBuffer;
17     public final int count;
18
19    public Reading1(byte[] sin,int count) {
20         this.count=count;
21         byte[] arr2=Arrays.copyOf(sin, sin.length);
22         this.tempBuffer =arr2 ;
23    }
24
25    public int getCount(){
26       return count;
27    }
28    //======================
29    public static byte[] Serialize(Reading1 r) {
30        try {
31            byte[] obj;
32            try (ByteArrayOutputStream baos = new ByteArrayOutputStream(2048)) {
33                ObjectOutputStream oos = new ObjectOutputStream(baos);
34                oos.writeObject(r);
35                oos.close();
36                obj = baos.toByteArray();       // get the byte array of the object
37            }
38            return obj;
39        } catch(Exception e) {
40            System.out.println("Can't serialize the voice packet.");
41        }
42
43        return null;
44    }
45    //======================
46
47       public static void main(String[] args) {
48       String str="vbfdvfhjjbj";
49
50      Reading1 p= new Reading1(str.getBytes(),10);
51      Reading1.Serialize(p);
52    }
53    }
```

Here we are using this class's function "Serialize(Reading1 r)" to serialize an object

The ByteArrayOutputStream class stream creates a buffer in the memory and all the data sent to the stream is stored in the buffer.

The ObjectOutputStream class is used to serialize an Object

PlayThread class

```
//************************************
    // recieving audio from ourserver and play it
    class PlayThread extends Thread {

          String token ="receive";
          PlayThread (String IpAddress,int receivingPort,String token){
              this.token=token;
          }

          byte tempBuffer[] = new byte[2100];
          int count;

          public void run() {
                      MulticastSocket server_socket;
               try {
                server_socket= new MulticastSocket(receivingPort);
                 add = InetAddress.getByName(IpAddress);
                     server_socket.joinGroup(add);
               }catch (IOException e) {
                e.printStackTrace();
                return ;
               }
                   System.out.println("token is "+token);
                   if(token.equals("send")){


                   final ScheduledExecutorService service = Executors.newSingleThreadScheduledExecutor();
                   service.scheduleWithFixedDelay(new Runnable() {
                        @Override
                        public void run() {
                            System.out.println("total received: " + count );
                            count = 0;
                        }
                   }, 5, 5, TimeUnit.SECONDS);

                  while (true){
                    try{
                       DatagramPacket receive_packet = new DatagramPacket(tempBuffer,tempBuffer.length);

                       server_socket.receive(receive_packet);

                       byte array[]=receive_packet.getData();

                       Reading1 f =DeSerialize.Deserialize(array);

                       byte array1[]=Arrays.copyOf(f.tempBuffer,f.tempBuffer.length);

                       count = f.getCount();

                       sourceDataLine.write(array1,0,array1.length);

                    }catch(Exception ee){
                       System.out.println("Exception2!"+ee);
                    }
                  }
              }
              else{
                 System.out.println("Can not play audio !");
              }
        }

      }

//************************************
}
```

Receiving and playing the audio from the server requires the use of different set of functions as in class PlayThread_ The datagram received from the server will be written to the temporary buffer at the receiver side and the audio will played using a thread. Here we use the DeSerialize class to deserialize the stream.

DeSerialize class

```
6 public class DeSerialize {
7     public static Reading1 Deserialize(byte[] tempBuffer) {
8         Reading1 r;
9 
10        try {
11            try (ByteArrayInputStream bais = new ByteArrayInputStream(tempBuffer);
12             ObjectInputStream ois = new ObjectInputStream(bais)) {
13               r = (Reading1) ois.readObject();
14 
15                return r;
16            }
17        } catch(IOException | ClassNotFoundException ex) {
18            System.out.println("Cann't deserialize the voice packet."+ex);
19        }
20 
21        return null;
22    }
23 }
```

The reverse process of creating object from sequence of bytes is called **deserialization**.
The ByteArrayInputStream class allows a buffer in the memory to be used as an InputStream. The input source is a byte array.

The **ObjectInputStream** class deserializes primitive data and objects previously written using an ObjectOutputStream. Following are the important points about BufferedInputStream:

- It is used to recover those objects previously serialized. It ensures that the types of all objects in the graph created from the stream match the classes present in the Java Virtual Machine.
- Classes are loaded as required using the standard mechanisms.

writeObject
Write the specified object to the ObjectOutputStream. The class of the object, the signature of the class, and the values of the non-transient and non-static fields of the class and all of its supertypes are written. Default serialization for a class can be overridden using the writeObject and the readObject methods. Objects referenced by this object are written transitively so that a complete equivalent graph of objects can be reconstructed by an ObjectInputStream. Exceptions are thrown for problems with the OutputStream and for classes that should not be serialized. All exceptions are fatal to the OutputStream, which is left in an indeterminate state, and it is up to the caller to ignore or recover the stream state.

AudioFormat getAudioFormat( ) ;

```java
// coding and format audio
private AudioFormat getAudioFormat() {
    float sampleRate = 16000.0F;
    int sampleSizeInBits = 16;
    int channels = 2;
    boolean signed = true;
    boolean bigEndian = true;
    return new AudioFormat(sampleRate, sampleSizeInBits, channels, signed, bigEndian);
}
```

One of the important parts of this chat voice application is to code and format the audio which the java code is presented below code part