Features “VoIP UserAgent”

UserAgent	
`static InetAddress getAddress()`	Return the loopback address
`static DatagramSocket getSocketOutgoing()`	Get the UDP Socket used to send packets to the mjUA on the port 5080. OUTPUT: • DatagramSocket used to send UDP packets
`static DatagramSocket getSocketIncoming()`	Get the UDP Socket used to receive packets from the mjUA on the port 5070. OUTPUT: • DatagramSocket used to receive UDP packets
`static void send(byte[] request)`	Send a request in byte to the server mjUA INPUT: • `request`: the Request to send, in byte
`static DatagramPacket listen()` `throws SocketTimeoutException`	Listen and return for a new DatagramPacket on the socketIncoming DatagramSocket. OUTPUT: • DatagramSocket received EXCEPTION: • `SocketTimeoutException` • `IOException`
`@Override` `void run()`	Run a new UserAgent inside a thread, send a Request and read the Response from the mjUA
Request	
`static String generateCallID()`	Generate and return a random CallID, a string of int values of 12 characters length.
`static String generateSenderTag()`	Generate and return a random Sender Tag,a string of int values of 12 characters length.
`static String generateBranch()`	Generate a pseudo-random branch of 13 characters length. Every branch must have as the first characters the sequence *z9hG4bK*, also known as magic number

`static void setSenderName(String newValue)`	Set the new UserAgent name
`static byte[] getInvite()`	Get the Invite Request, in byte
`static byte[] getAck()`	Get the Ack Request, in byte
`static byte[] getBye()`	Get the Bye Request, in byte
Response	
`static void setResponsePacket(DatagramPacket packet)`	Set the new Response Packet and the new message received.
`static void showMessage()`	Show the UserAgent the message received from the mjUA and a few lines of description.
Session	
`static void setActive(boolean value)`	Set the status of the current Session. INPUT: • `TRUE` the Session is active • `FALSE` the Session is not active
`static boolean isActive()`	Get the status of the current Session
`static void addRequest(byte[] newRequest)`	Add a new Request (in byte) to the Request list.
`static void addResponse(byte[] newRequest)`	Add a new Response (in byte) to the Response list.
`static void addPacket(DatagramPacket newPacket)`	Add a new Packet UDP, sent or received, to the Packets list.
`static void clear()`	Clear all the lists in the Session.
`static void saveRequestFile(String request, String name)`	Save the Request sent on a file in `resources/requests` INPUT: • `request:` the string to save on a file • `name:` name of the file
`static String logsMessage()`	Return a string containing every packet's data in the Packets list. OUTPUT: • the Session's logs message

`static void save()`	Save the Session's logs message in a external file named logs.txt in `resources/requests`
RTP Packet	
`byte[] getHeader()`	Get the header of the RTP Packet.
`void incrementSequence()`	Increment the Sequence value by one unit.
`void incrementTimestamp()`	Increment the Timestamp value by 160 unit.
Output Audio	
`static void setSourcePort(int newValue)`	Set the RTP Port to which send the RTP Packets.
`static void setSendingAudio(boolean newValue)`	Set the status of the SendingValue attribute: INPUT: • `TRUE` if the UserAgent is sending audio • `FALSE` if the UserAgent is not sending audio
`static boolean isSendingAudio()`	Get the status of the SendingAudio attribute
`static void sendAudio(byte[] toSend)`	Send a RTP Packet (in byte) through Socket to the mjUA. INPUT: • `toSend` the RTP Packet, in byte.
AudioThread	
`static DatagramSocket getSocketIncoming()`	Get the UDP Socket used to receive packets from the mjUA on the port 4070. OUTPUT: • DatagramSocket used to receive RTP packets
`@Override` `void run()`	Start a thread that in a loop receives the RTP packets sent by the mjUA (which simple contains silence), edit some of the 160 bytes of the payload with random values and send it back to the mjUA, which receives and play it.The sound will result in a spoiled noise.
AudioSinusoidalThread	
`static double getFrequency()`	Return the value of the Frequency

`static double getAmplitude()`	Return the value of the Amplitude
`static double getTimeIncrementation()`	Return the value of the Time Incrementation
`static void setFrequency(double newValue)`	Set the new value of Frequency attribute INPUT: • `newValue` of Frequency (in double)
`static void setAmplitude(double newValue)`	Set the new value of Amplitude attribute INPUT: • `newValue` of Amplitude (in double)
`static void setTimeIncrementation(double newValue)`	Set the new value of Time Incrementation attribute INPUT: • `newValue` of Time Incrementation (in double)
`@Override` `void run()`	Start a thread that create a new RTP Header and in a loop calculate a mathematical sine wave of given frequency and amplitude, as a function of x. The wave increments x by a given value. The sinusoid calculated is then compressed using the `G711.linear2ulaw` of `sip.jar library`, and fills the entire bytes of the RTP body. At the end of every loop, send the RTP Packet, and increments the Sequence and the Timestamp values. A RTP Packet is sent every 20ms.
AudioFileThread	
`@Override` `void run()`	Start a thread that create a new RTP Header and take a `.wav file` in the `resources/audio` folder to send it to the mjUA. It calculates the number of packets to be created fitting the entire audio file in different RTP's body of 160 bytes length, then start creating a loop that insert the AudioInputStream in every RTP body, and send the packet to the mjUA. Once a packet is sent increments the Sequence and the Timestamp value and send the next one after 20ms.