**PEER-TO-PEER MULTICAST TEXT CHAT SYSTEM**

**Introduction:**

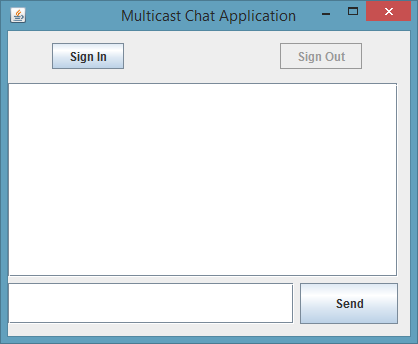
Peer-to-peer multicast text chat system is the process of sending and receiving data packet or message packet to/from machines or systems connected on a same network. It uses Internet layer, Transport Layer and Application Layer of a machine built on TCP/IP architecture. In this application all clients communicate as peers.

**Project Design:**

The application developed is a distributed application, in which the machines connected in a network communicate by sending the messages. The application runs on java platform

**GUI Design:**

NetBeans JFrame form is used for the design of Graphical User Interface. In the GUI of Chat Application developed, we have a text area, text field, send button, sign in and sign out buttons. Sign In button is used to connect to the Group chat. Text Area to display the text messages sent from other users and also to display the messages sent by the user. User uses the Text field to enter the text message. Send button to send the text entered in the text field. Sign out button to disconnect from the group chat.



**Implementation:**

Application is developed on java platform. And the concept of nested class is used. Outer class extends JFrame for creation of GUI and also includes sending the packets to users in the group chat. Inner class is a receiver class, where the user receives the packets from the group chat. In order to use the text area field of JFrame and to implement threading, this class is placed within another class.

Once the java code is run, the main method of the outer class is invoked. Here the object of outer class is invoked and its constructor is invoked. Inside the constructor method, all the JFrames components, text field, text area, send, sign in and sign out buttons are initialized.

When the user clicks on Sign In button, signInButtonActionPerformed method is invoked, in which connection to the specified port is made. Firstly, InetAddress of the Multicast group is taken. And then port number of the Multicast is received. Thereafter, multicast socket is created and the user is made to join the group (to the InetAddress received) to start sending and receiving the packets.

Once the connection is established user is ready to send and receive the Datagram packets. In order to receive and send the packets simultaneously, the concept of threading is used. As soon as the connection is established, receiver thread is initialized and run method of the receiver class is executed. In the run method, DatagramPacket sent from the members of group chat is received and displayed in the user’s text area

|  |
| --- |
| **Business logic for sending DatagramPacket** |
| try {  // Receive the text entered by user  String userText = textField.getText();  // DatagramPacket object creation for sending the packet  DatagramPacket sendDatagramPacket = new DatagramPacket(userText.getBytes(), 0, userText.length(), group, portNumber);  //Send the packet to MulticastSocket  chat.send(sendDatagramPacket);  // To print the user message in text area  textArea.append(InetAddress.getLocalHost().getHostName() + ": " + userText + "\n");  // To wrap the text printed in text area  textArea.setLineWrap(true);    // To set the text field back to blank  textField.setText("");  } catch (IOException ex) {  System.out.println(ex);  }  } |

User enters the text in the text field. And to send this message to the group chat, user has to hit enter key on the keyboard or press send button beside the text field. The entered text is made to append to text area to display it back to the user. And the process of sending and receiving the packets is continued.

|  |
| --- |
| **Business logic for Receiving DatagramPacket** |
| try {  while (true) {  byte receiveBuffer[] = new byte[1000];  //Creating DatagramPacket object to receive the packet  DatagramPacket receiveDatagramPacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);  //Receive the DatagramPacket  chat.receive(receiveDatagramPacket);  //to make sure USER MESSAGE not to receive again  boolean result = InetAddress.getLocalHost().getHostAddress().equals(receiveDatagramPacket.getAddress().getHostName());  if (!result) {  String receivedMessage = new String(receiveDatagramPacket.getData()).trim();  //To print the message in text area  textArea.append(receiveDatagramPacket.getAddress().getHostName() + ": " + receivedMessage + "\n");  }  }  } catch (IOException ex) {  System.out.println(ex);  } |

If the user wishes to sign out from the group chat, sign out button is clicked, then signOutButtonActionPerformed method is invoked. Sign out button generates another DatagramPacket and sends it to users in group chat saying the user has left the group. And the connection is closed.

**Outputs:**

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Sign In and regular message at user end | Sign in, sign out and regular message at Receiver | Sign out message at user |

**Testing:**

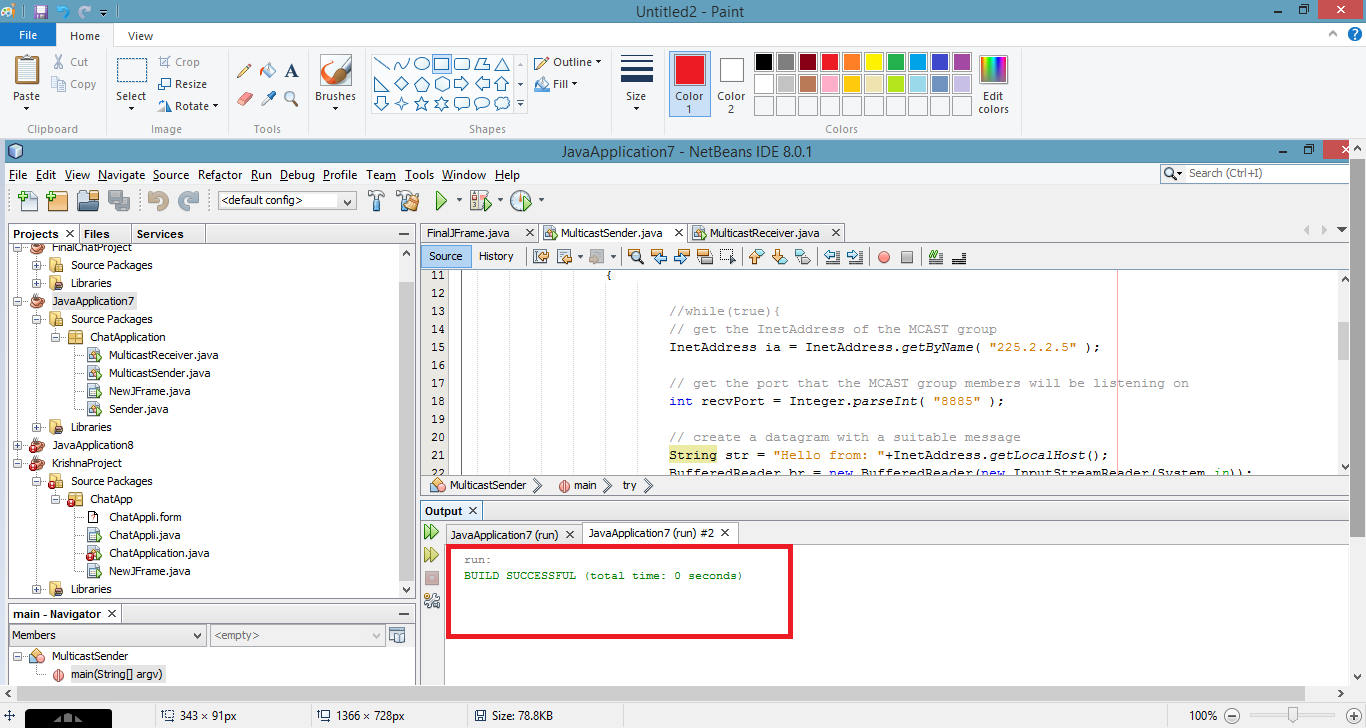
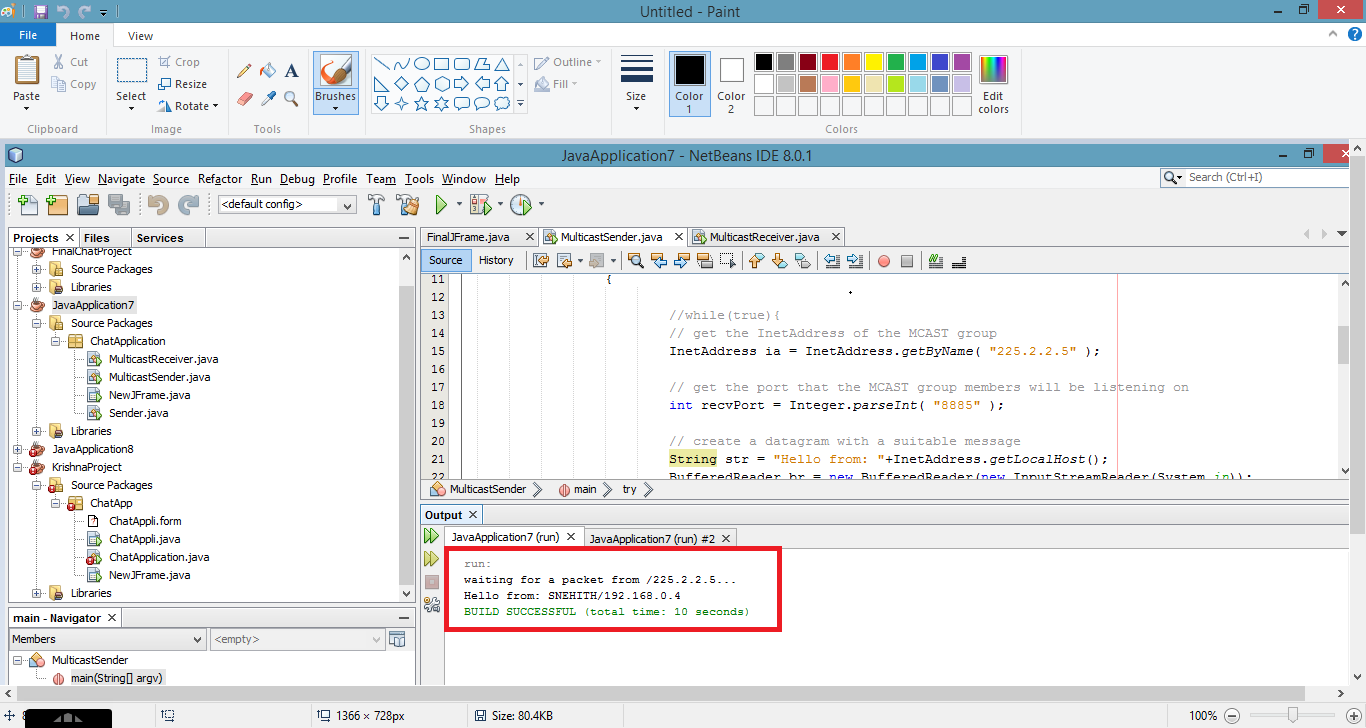
1. Verified whether sign out button works before signing into the group chat.
2. Verified whether sign in button works before signing out of group chat.
3. Verified whether user can send the message before user signs into the group chat.
4. Verified whether user can send the message to group successfully.
5. Verified whether user can receive the messages from the members of the group chat.
6. Verified whether send button can send the text entered in text field successfully
7. Verified whether enter key on keyboard can send the text entered in text field successfully
8. Verified whether sign in message is displayed to user.
9. Verified whether sign out message is displayed to user.
10. Verified multicasting by running the same chat application of different machines connected on same network.
11. Verified whether sign in button works again or not once the user signs into chat application.
12. Verified whether sign out button works again or not once the user signs out of chat application.

**Tasks:**

**Task 1:**

**Compile and run the two applications. Do they behave as expected?**

When the two applications (MulticastSender and MulticastReceiver) are run, the MulticastSender just sends a packet and MulticastReceiver receives the same packet. They behave as expected, that is the MulticastSender sends “Hello from“ message and the MulticastReceiver receives that message.

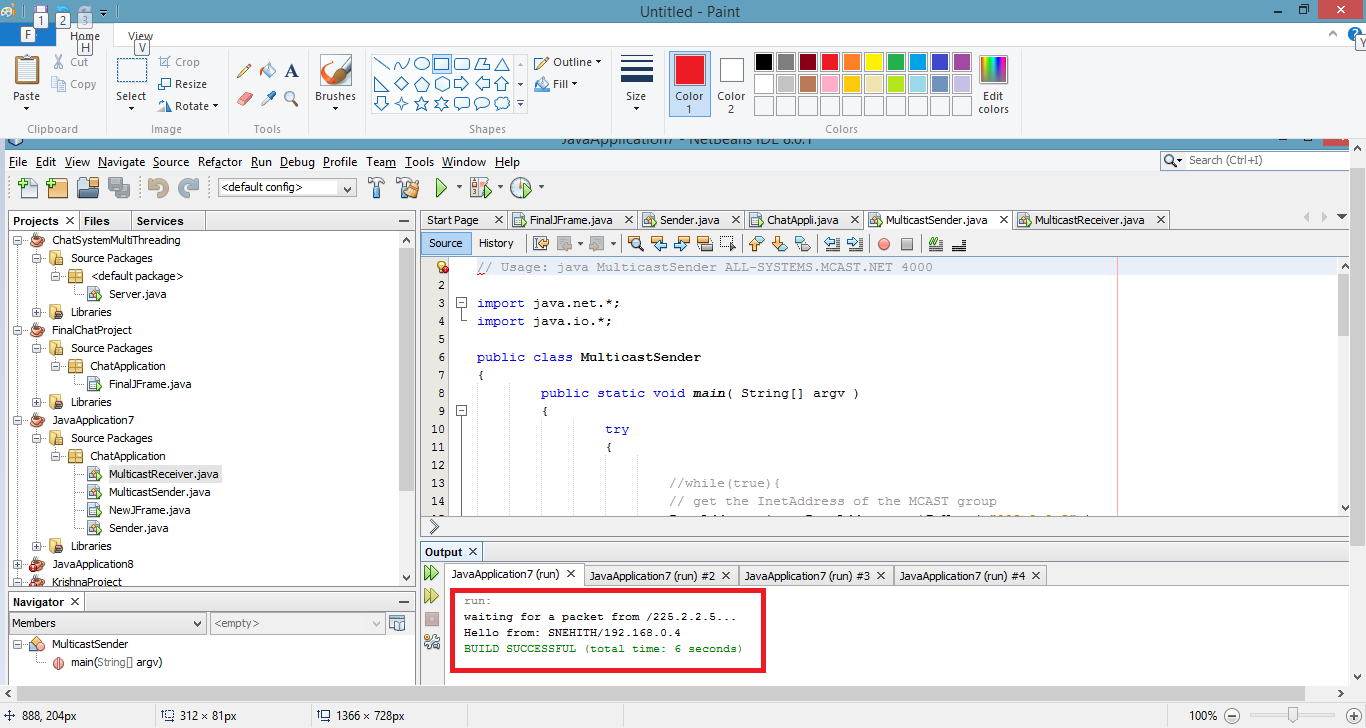
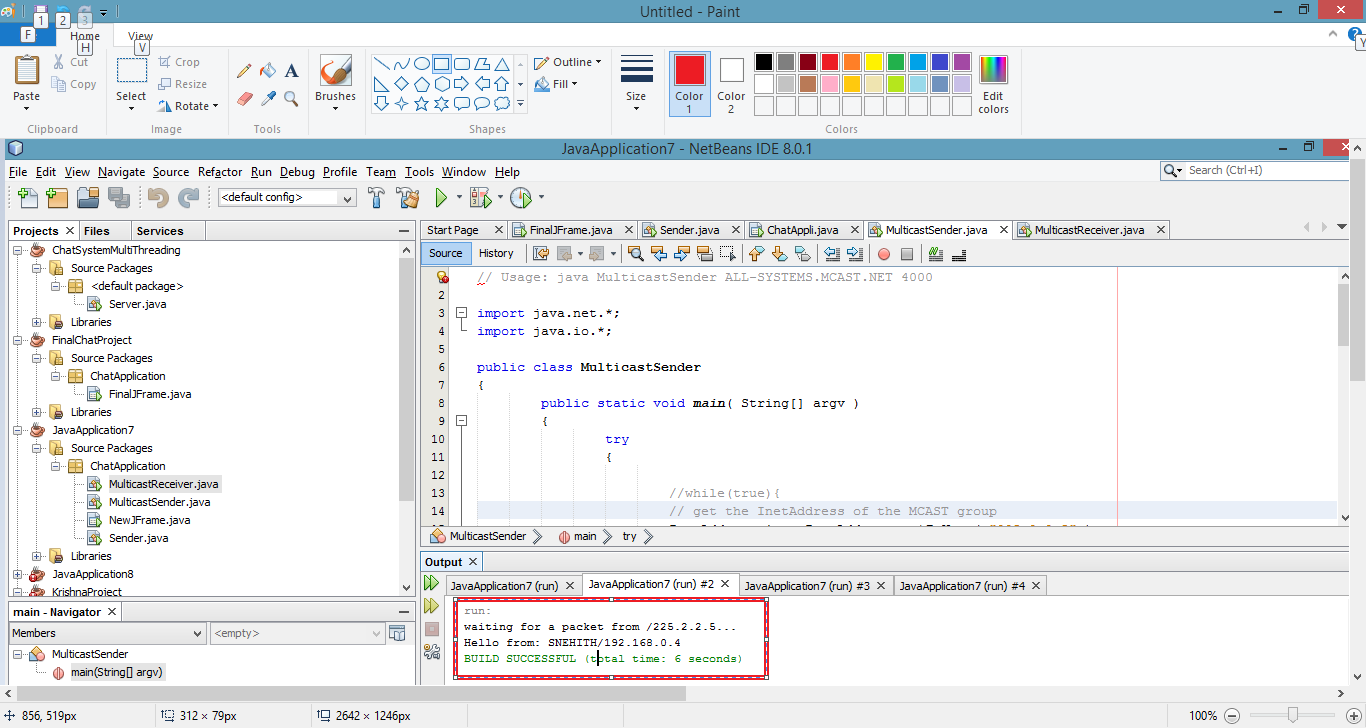
 

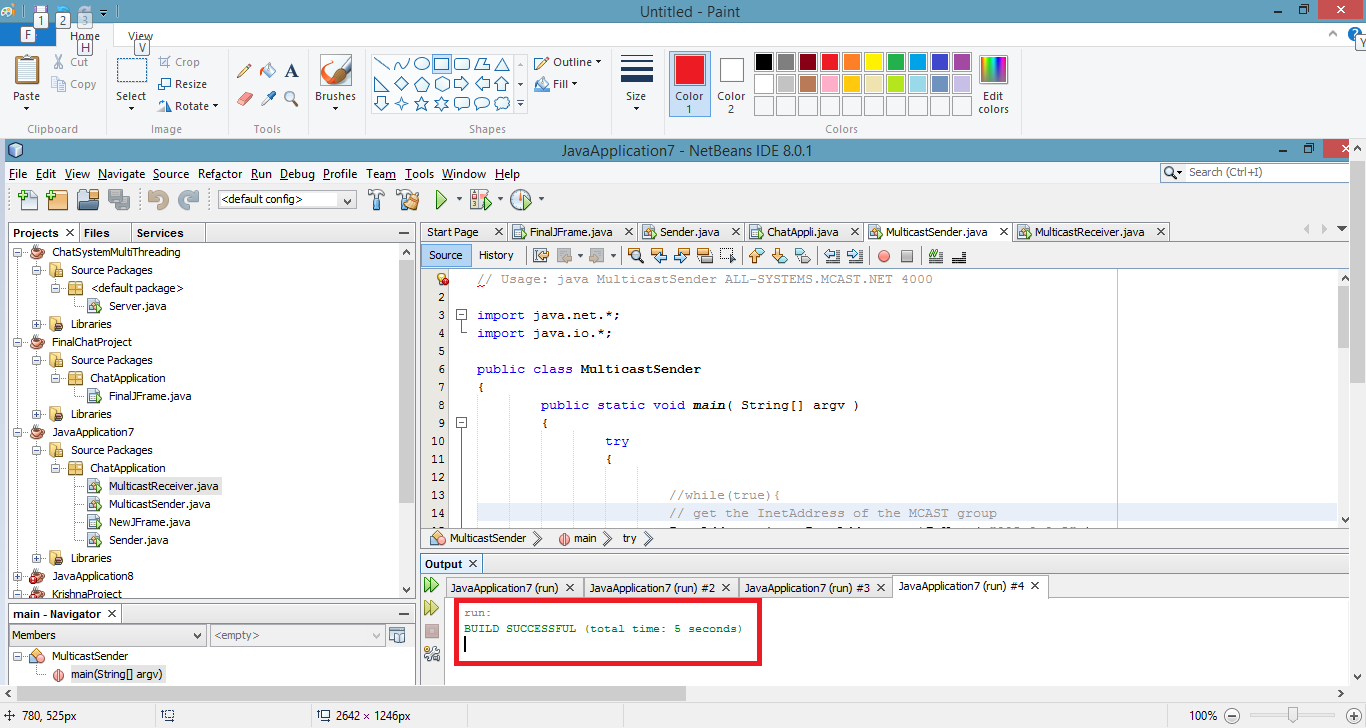
MulticastSender Output MulticastReceiver Output

**Task 2:**

**Run multiple instances of the MulticastReceiver on the same machine to check that multicasting is indeed occurring.**

When multiple instances of MulticastReceiver are run on same machine, all the Receivers receives the message sent from the MulticastSender. As all the MulticastReceivers are connected to same socket of a network, all MulticastReceivers receives the packet from MulticastSender.

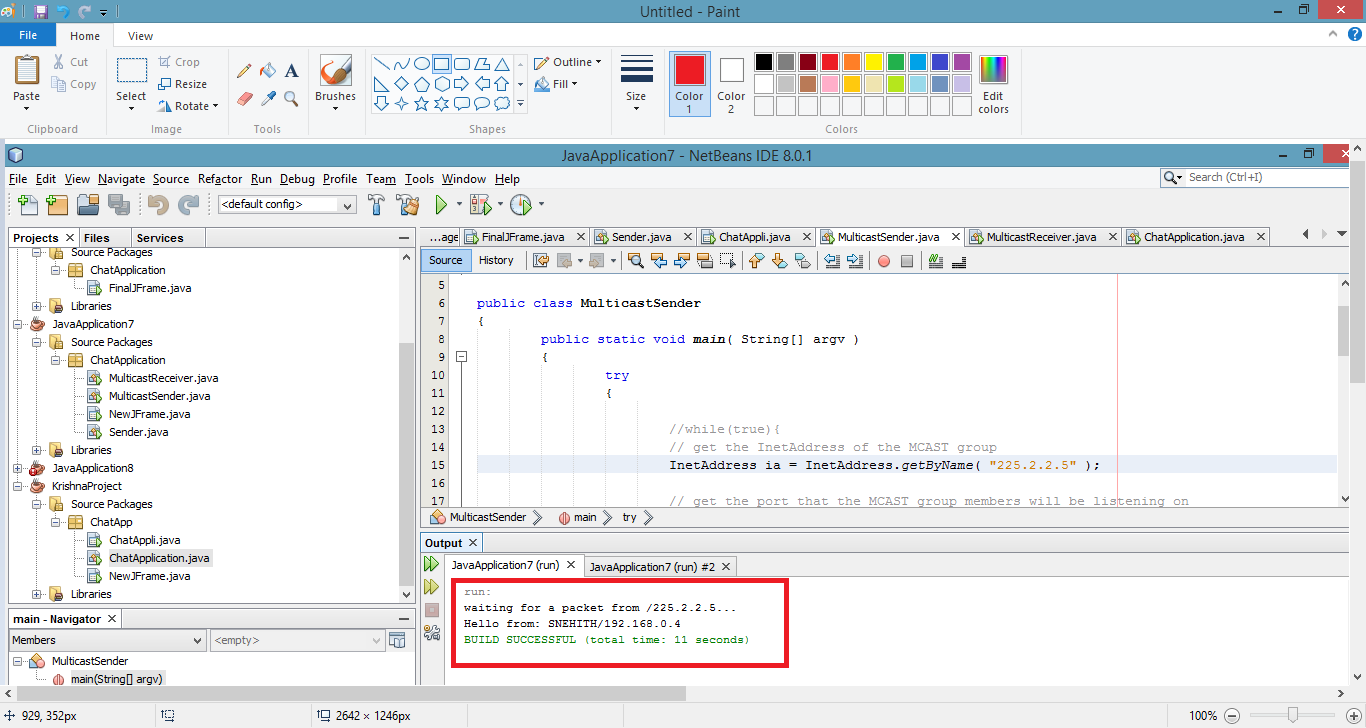
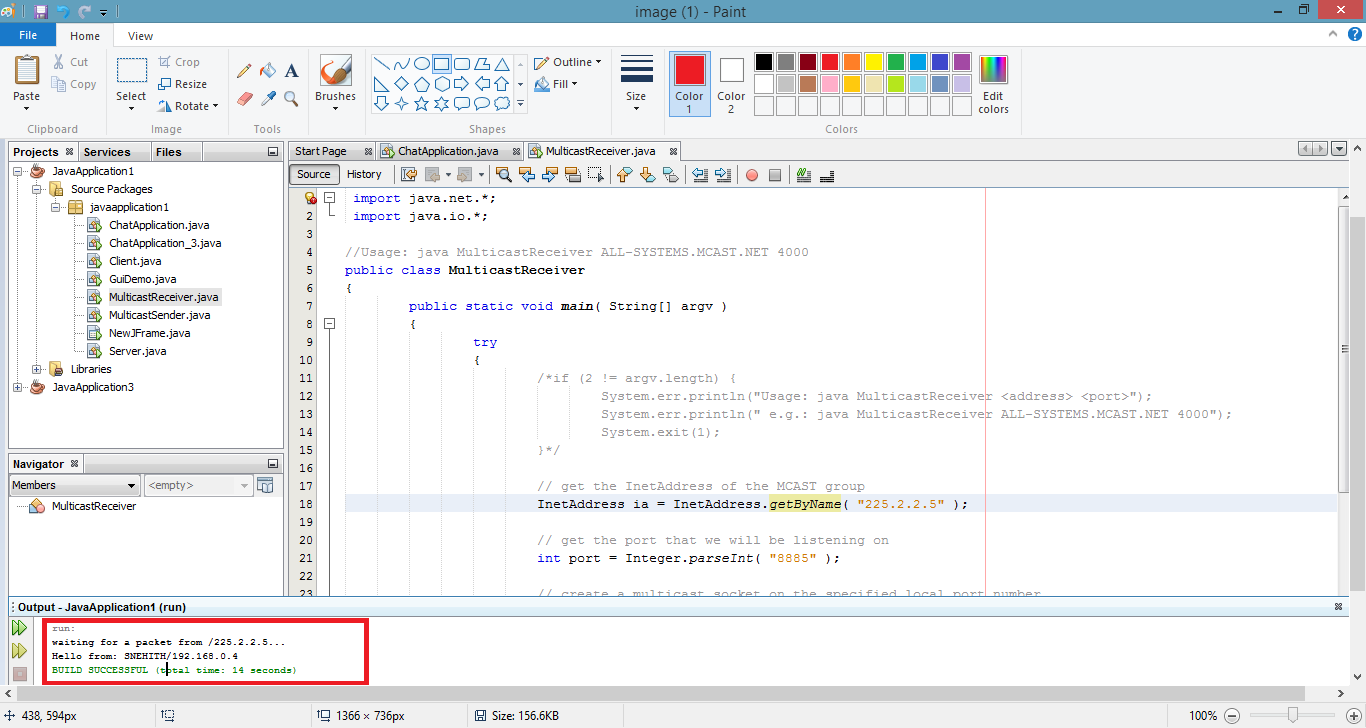
MulticastReceiver run1 output MulticastReceiver run2 output 

MulticastSender output

**Task 3:**

**After this, try running instances of the receiver on different machines - try with adjacent machines in the lab.**

When instances of receiver is run on different machines, output of the receiver will be the Hello from message and the IP address of the sender machine.

Output in machine 1 Output in machine 2

**Task 4:**

**An interesting, though very annoying, effect might occur if the entire class were attempting this tutorial exercise at the same time - what would this effect be?**

It would result in NL-Network-01 error according to neotys community (http://answers.neotys.com/questions/19544-what-does-nl-network-01-errors-mean-and-what-can-i-do-to-troubleshoot-the-cause).

**Task 5:**

**How could you modify the receiver program so that it doesn't quit every time it receives a packet as needed in Part 2?**

When try catch block in MulticastReceiver code is looped in an infinite while loop, it will not quit every time it receives a packet.

**Task 6:**

**How could you modify the sender program so that it can accept text messages from the keyboard inputs, and doesn’t quit every time it sends a packet, as needed in Part 2?**

In MulticastSender code if BufferedReader is used, user can text the messages from keyboard inputs. If user doesn’t want to quit after sending the packet, include the try catch block in MulticastSender code in an infinite while loop.

**References:**

1. <http://www.tutorialspoint.com/java/java_multithreading.htm>
2. <http://download.java.net/jdk7/archive/b123/docs/api/java/net/MulticastSocket.html>
3. <http://stackoverflow.com/>
4. <http://en.wikipedia.org/wiki/Distributed_computing>
5. <http://answers.neotys.com/questions/19544-what-does-nl-network-01-errors-mean-and-what-can-i-do-to-troubleshoot-the-cause>