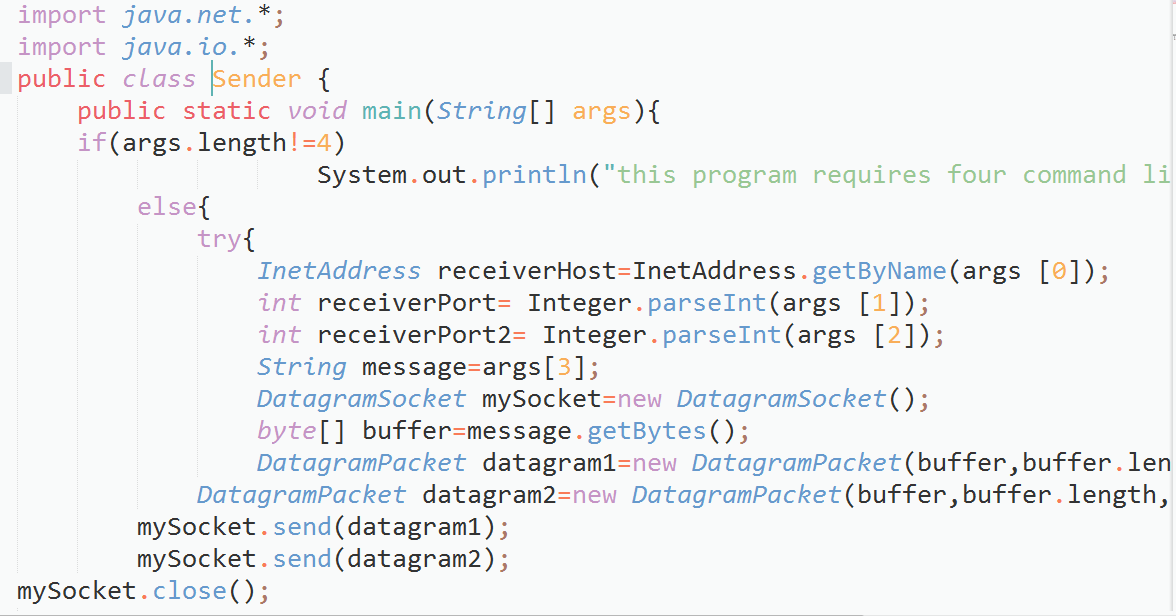
**Lab #01**

**Objective: To become familiar with Socket programming.**

**Task 1:** Modify the sample code so that the sender uses the same socket to send the same message to two different receivers. Start the two receivers first, then the sender. Does each receiver receive the message? Capture the code and output. Describe the outcome.

**Sender Class:**

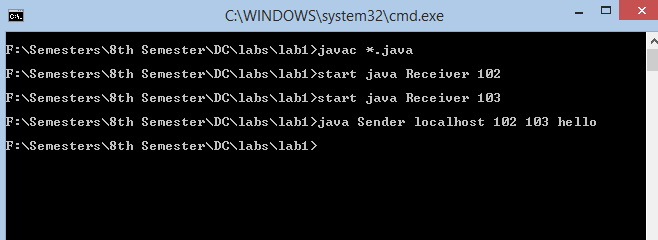
****

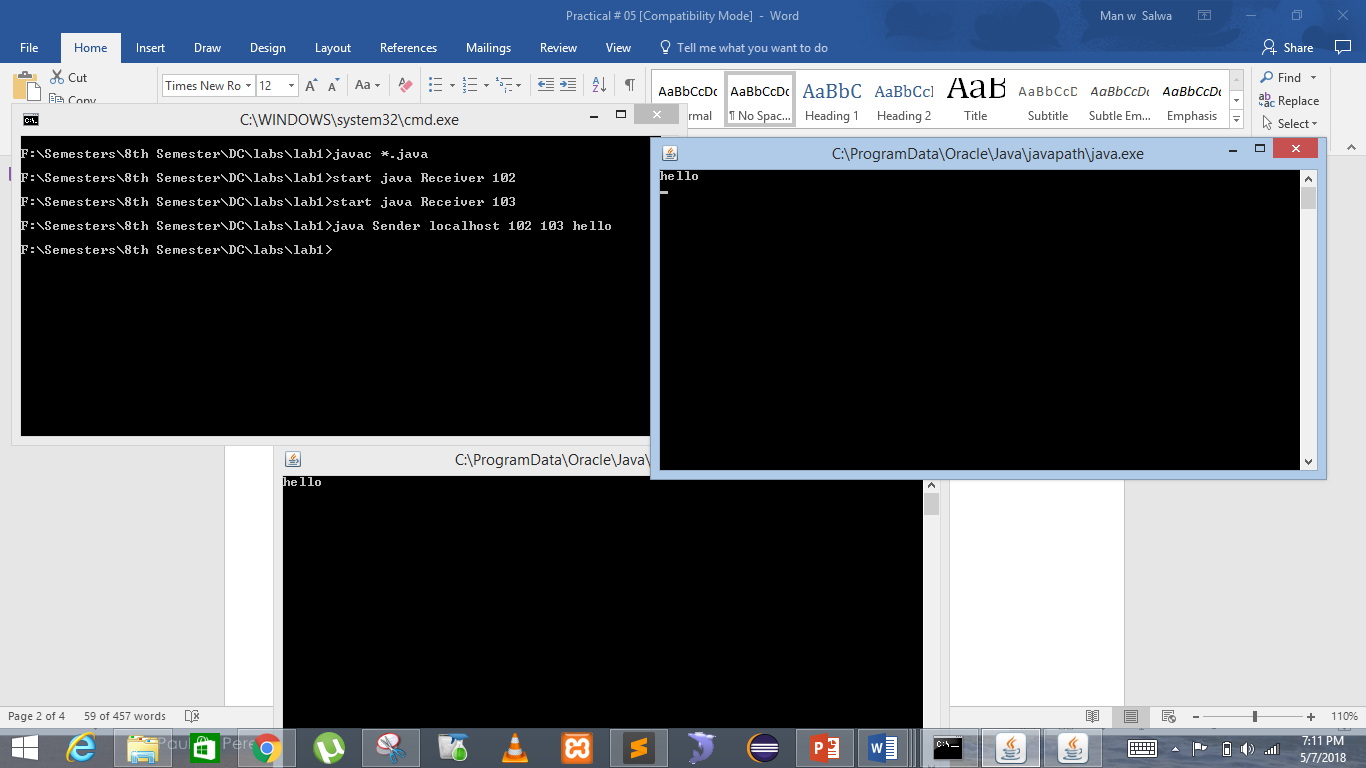
16-15SW38

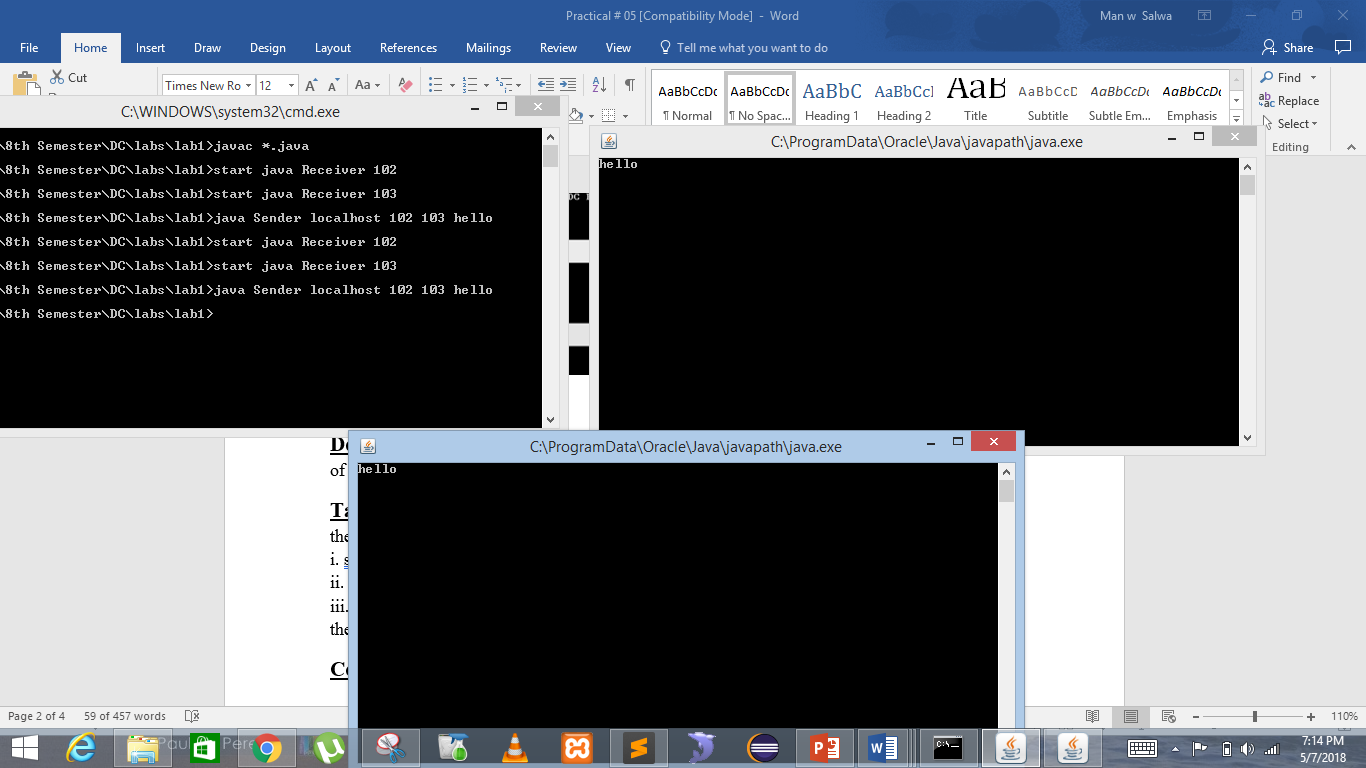
**Receiver Class:**

****

**Output:**

****

****

****

**Task 2:** Modify the sample code so that the receiver loops five times to repeatedly receive then display the data received. Recompile. Then  
i. start the receiver  
ii. Execute the sender, sending a message “message1”, and  
iii. In another window, start another instance of the sender, sending a message “message2”. Does the receiver receive both the messages? Capture the code and output.

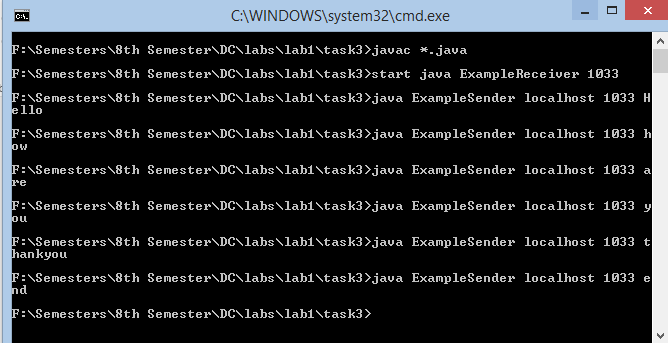
**Example Sender:**

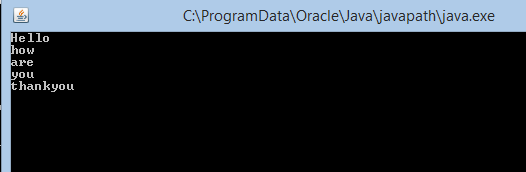
****

**Example Receiver:**

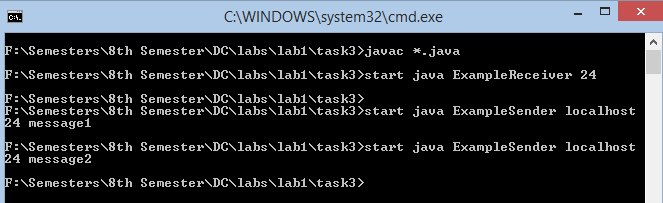
****

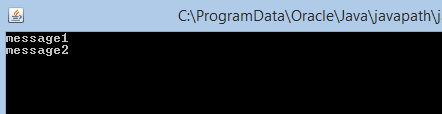
**Output:**

****

****

**After re-compiling:**

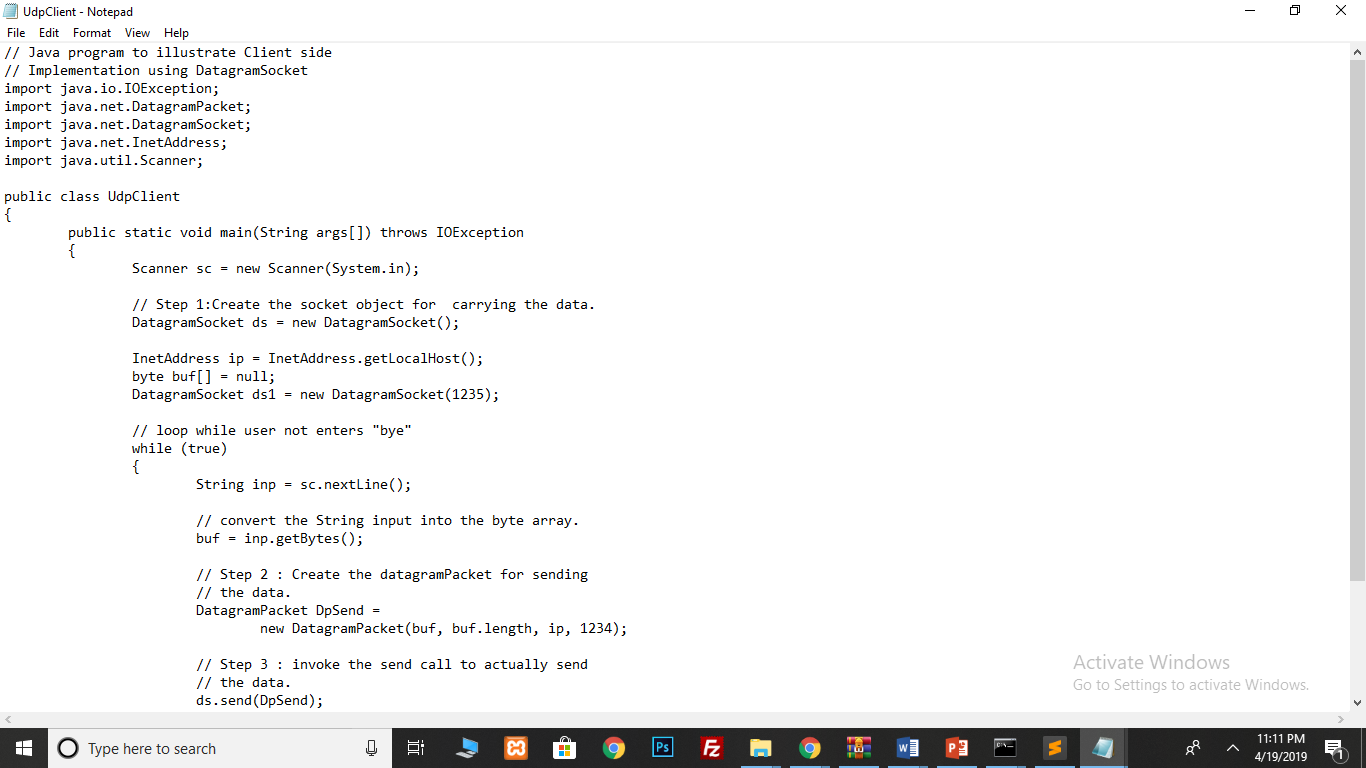
****

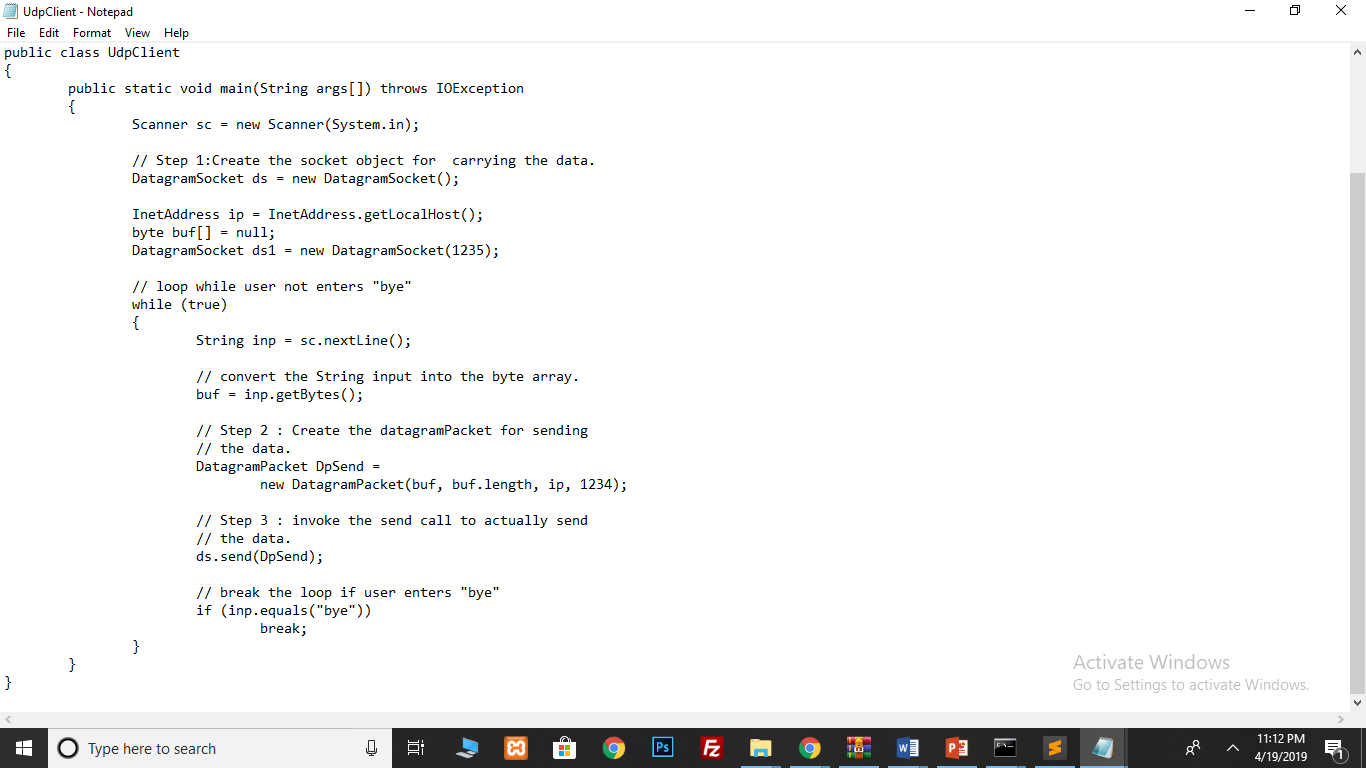
****

**Task3:** Modify the sample code to cater to a two-way communication i.e. Sender sends a message to the Receiver, the Receiver receives the message and sends a reply to the Sender in return.

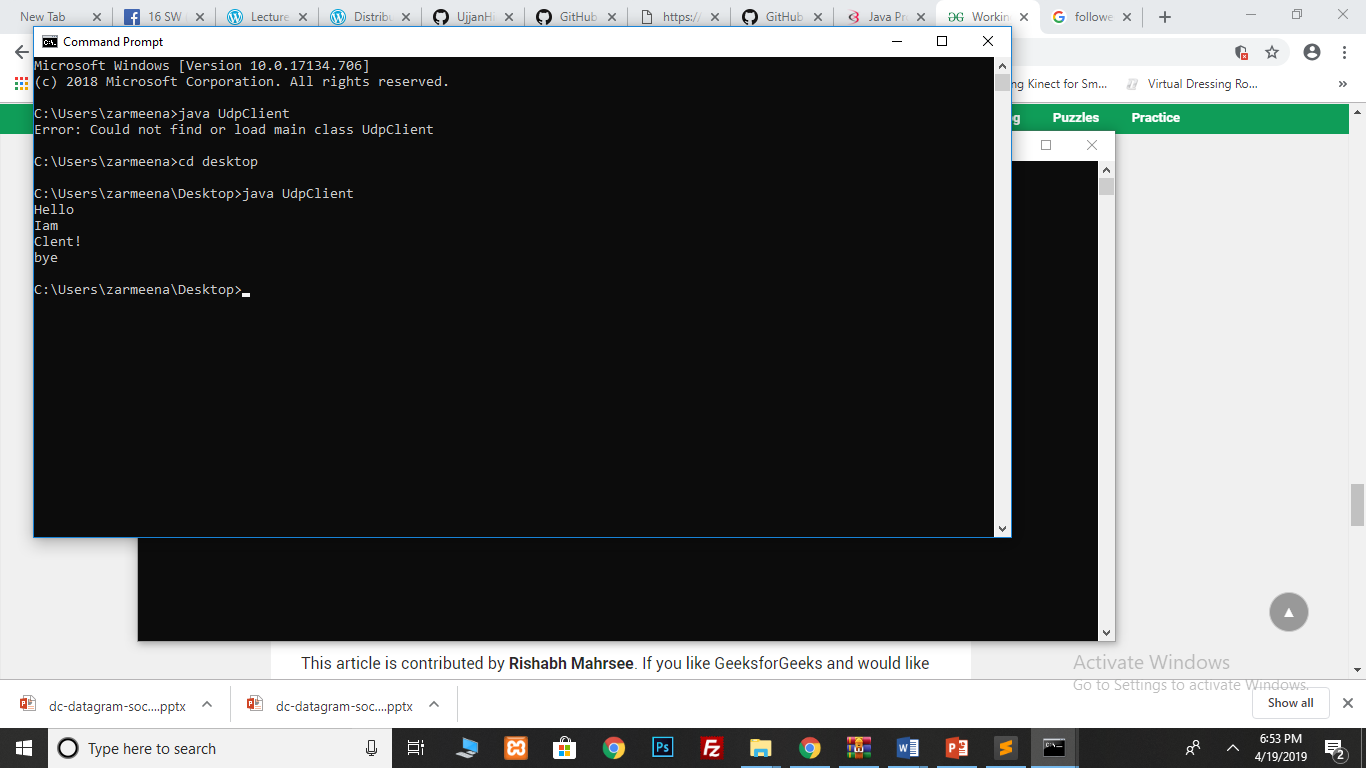
**Client is sending, and server is receiving:**

**Client-Side Implementation**

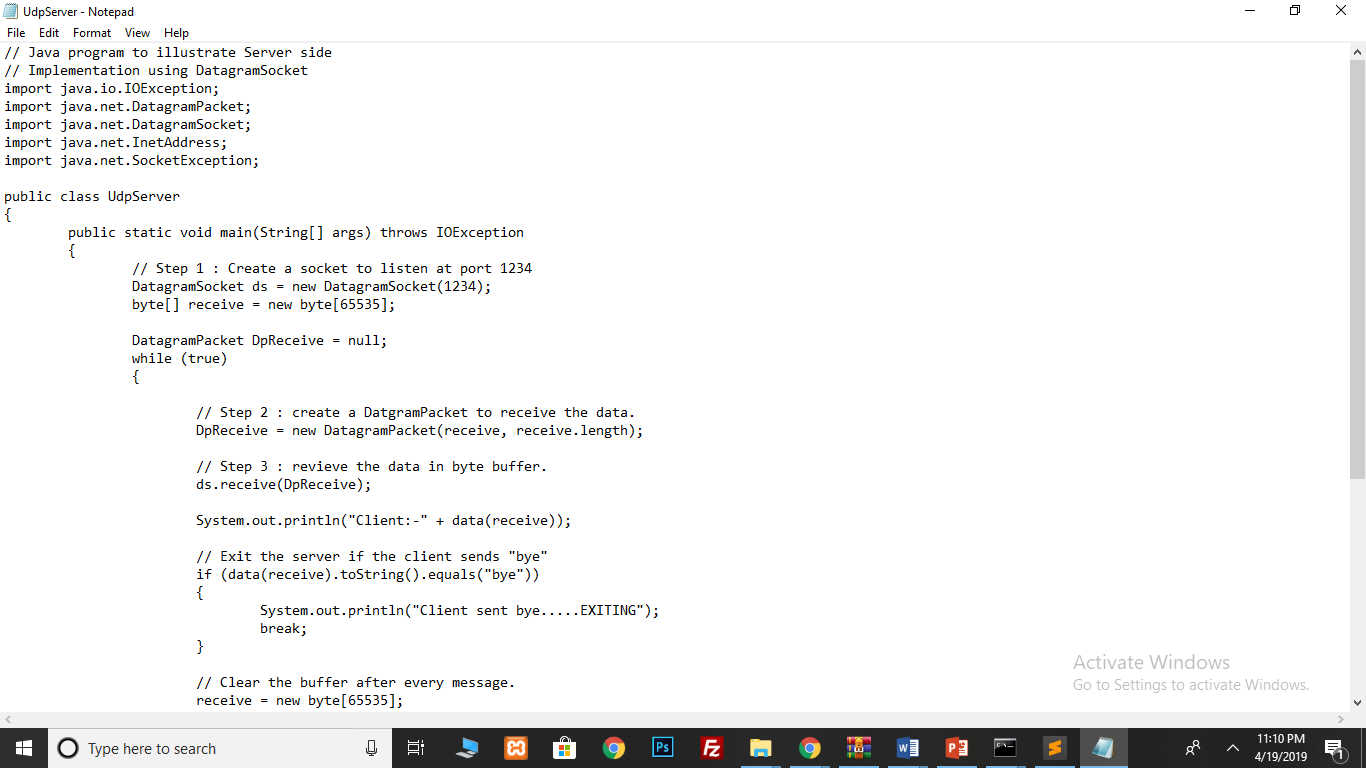


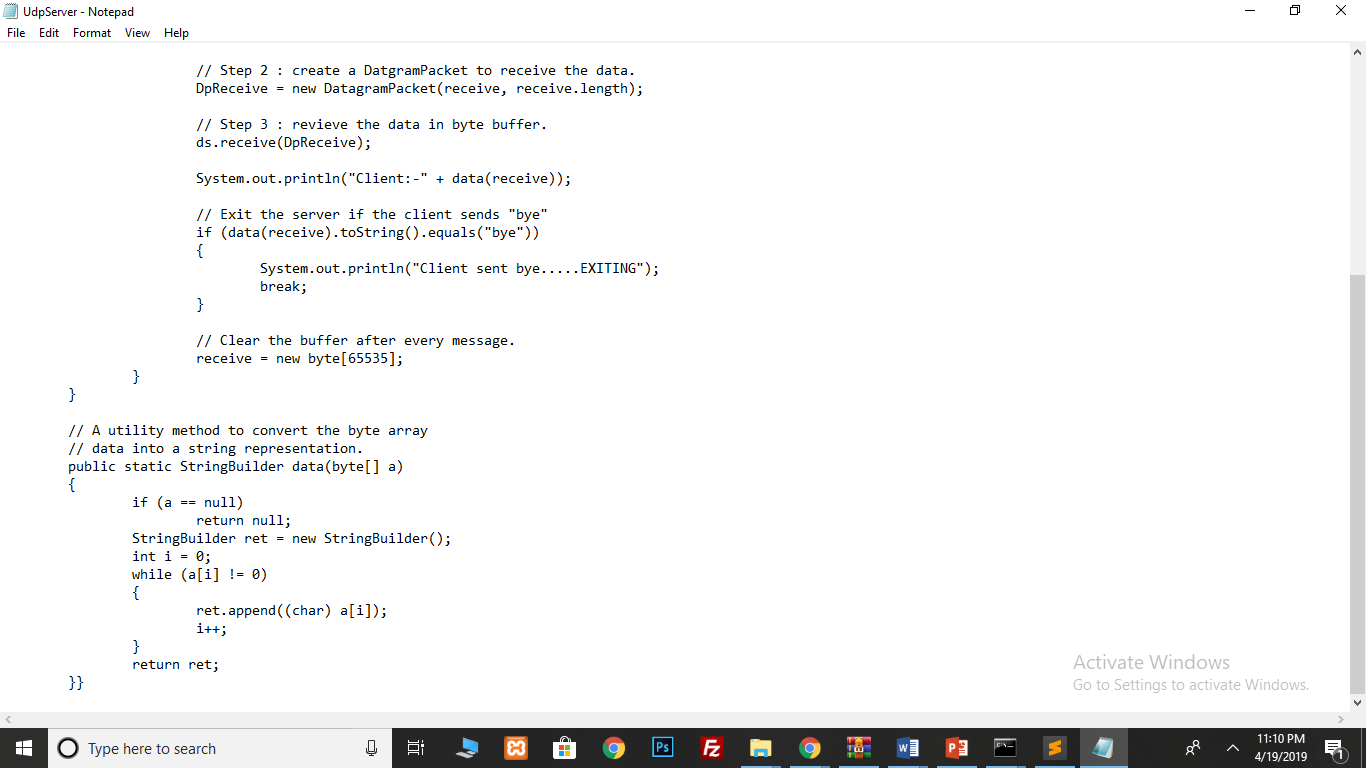


**Output**

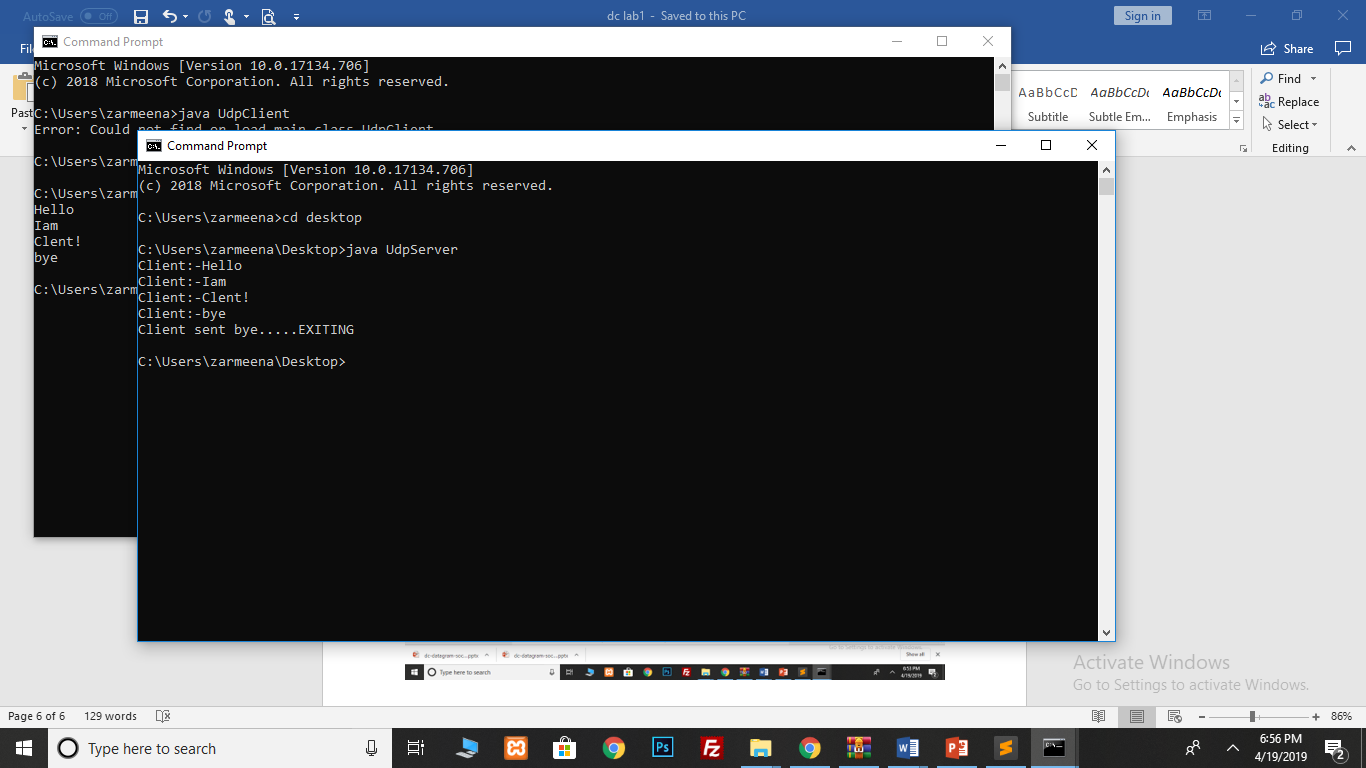


**Server-Side Implementation**





**Output**



Note: - In order to test the above programs on the system, please make sure that you run the server program first and then the client one. Make sure you are in the client console and from there keep on typing your messages each followed with a carriage return. Every time you send a message you will be redirected to the server console depending on your environment settings. If not redirected automatically, switch to server console to make sure all your messages are received. Finally, to terminate the communication, type "bye" (without quotes) and hit enter.

**Task 4:** Implement two simple programs using Java datagram sockets, which broadcasts and multicast your roll number to all or selected network nodes respectively.

**UDP Multicast**

Java uses MulticastSocket class to create UDP multicast sockets to receive datagram packets sent to a multicast IP address.

A multicast socket is based on a group membership. After creating and bounding a multicast socket, call its joinGroup(InetAddress multiCastIPAddress) method to join the multicast group, any datagram packet sent to that group will be delivered to this socket.

To leave a group, call the leaveGroup (InetAddress multiCastIPAddress) method.

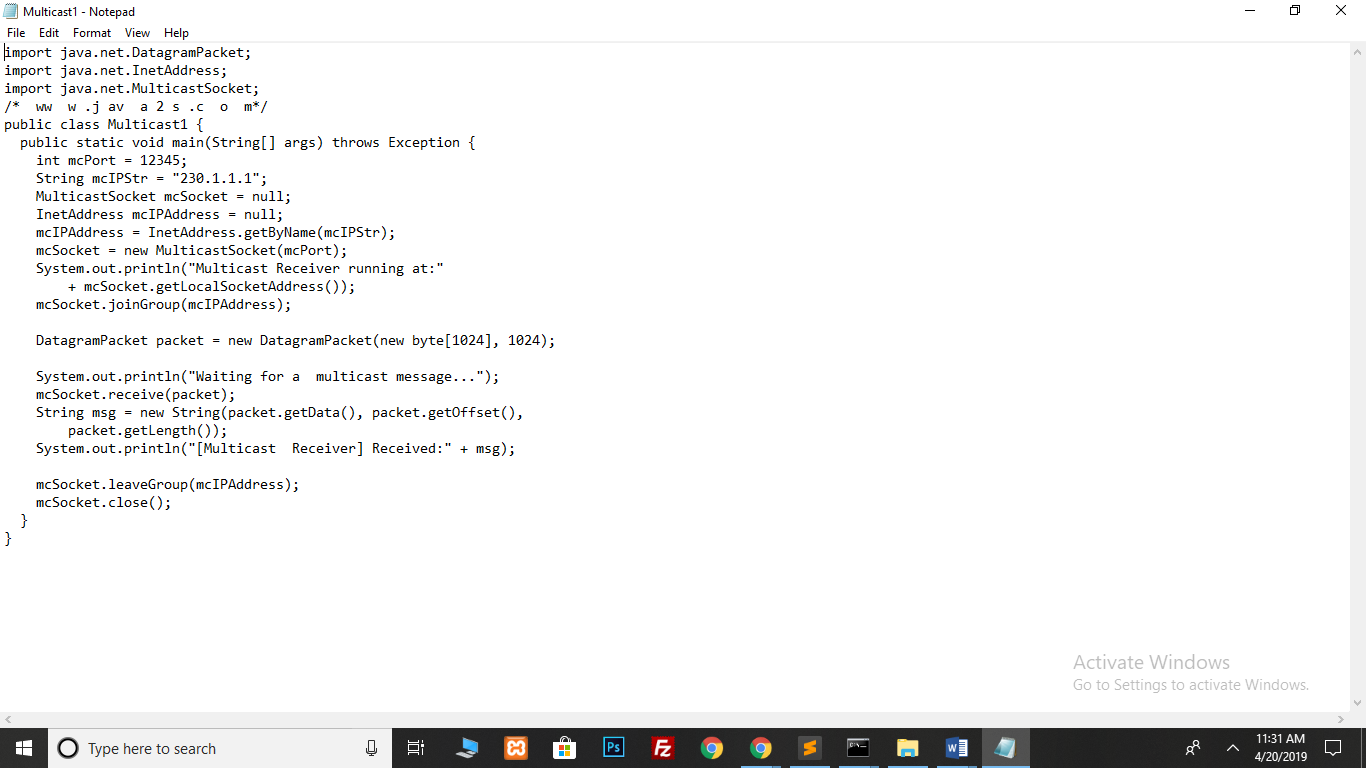
In IPv4, any IP address in the range 224.0.0.0 to 239.255.255.255 can be used as a multicast address to send a datagram packet.

The IP address 224.0.0.0 is reserved and you should not use it in your application.

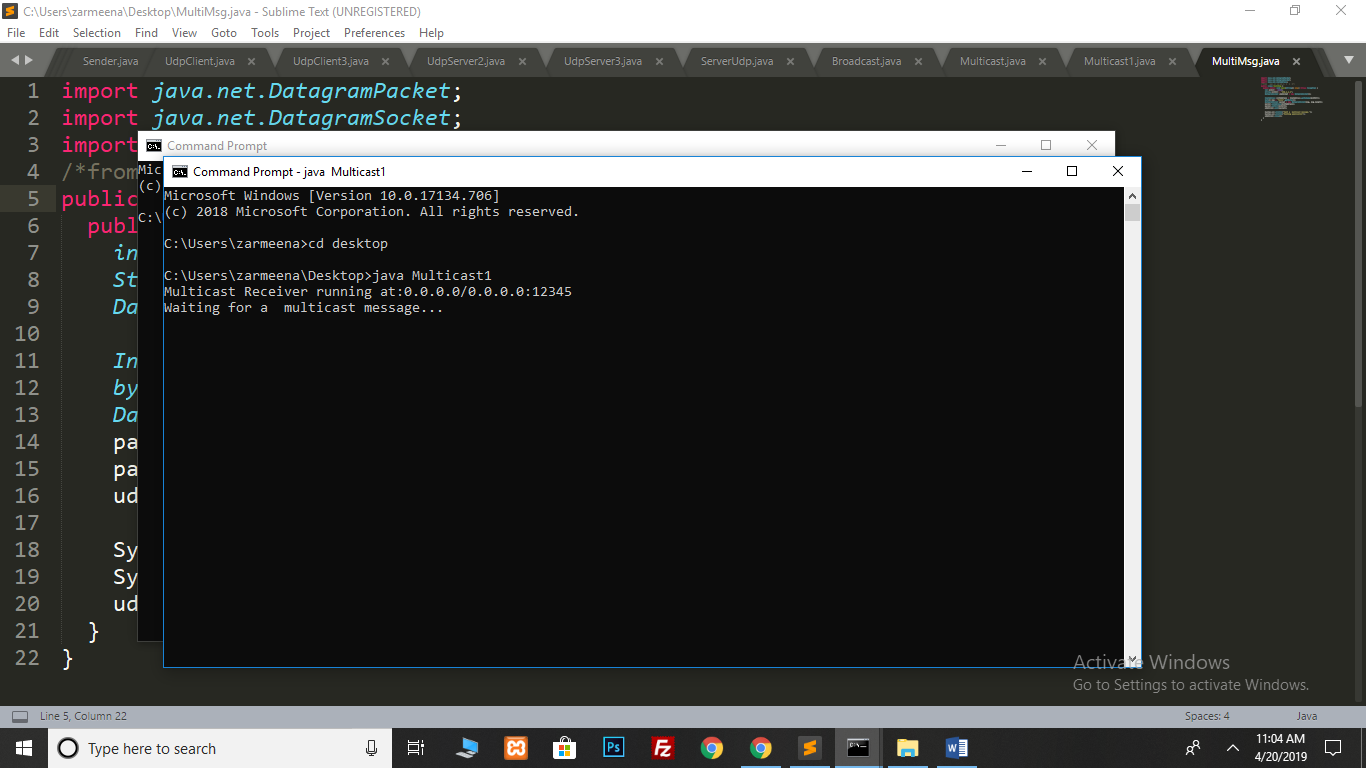
A multicast IP address cannot be used as a source address for a datagram packet.

**Code**

A UDP Multicast Socket That Receives UDP Multicast Messages.

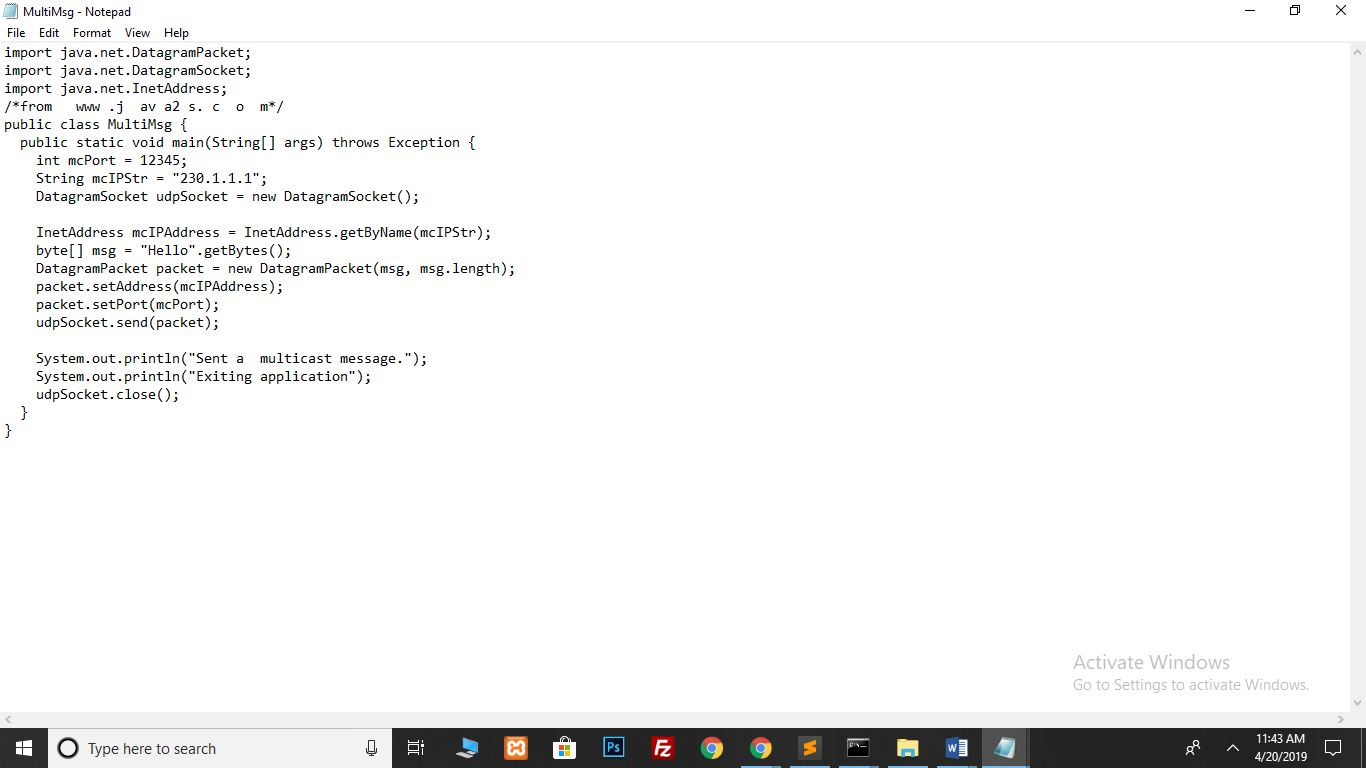


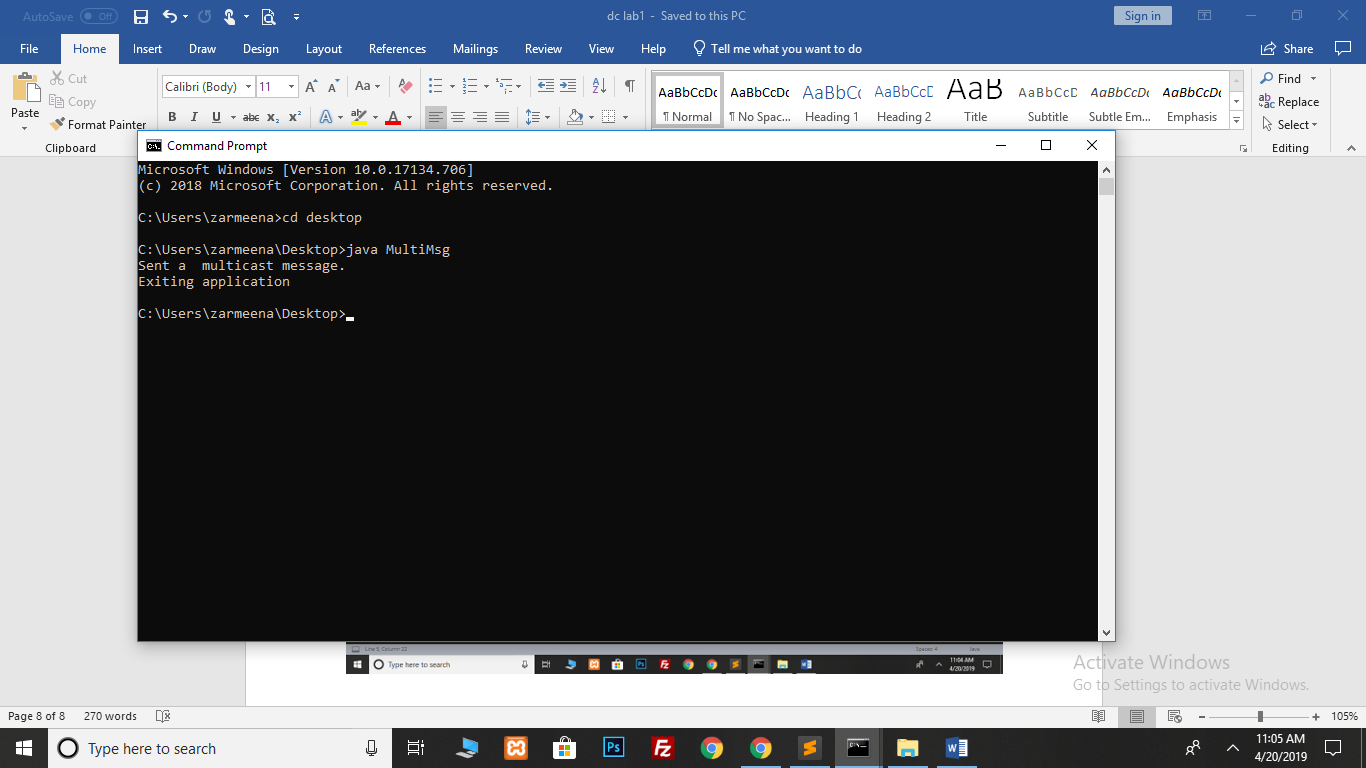
**Output**

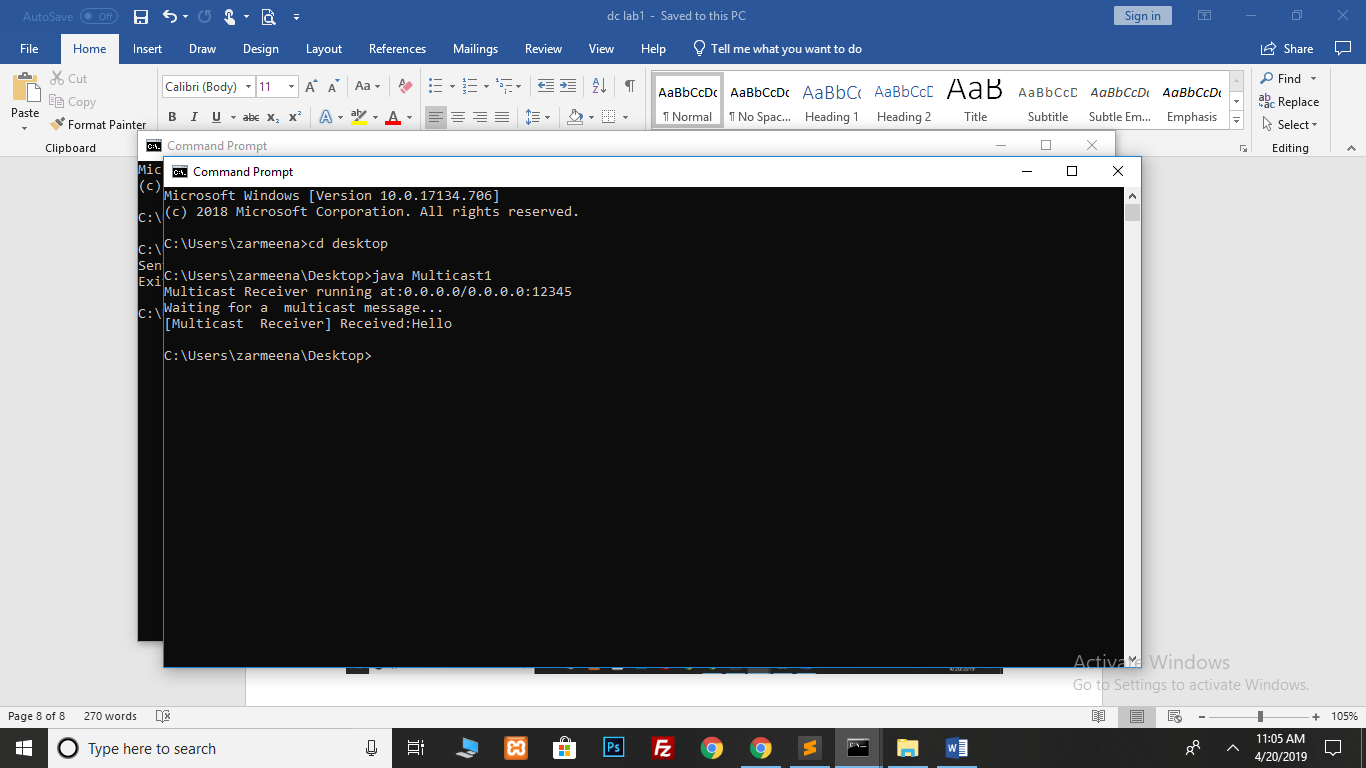


**Code**

A UDP Datagram Socket, a Multicast Sender Application.



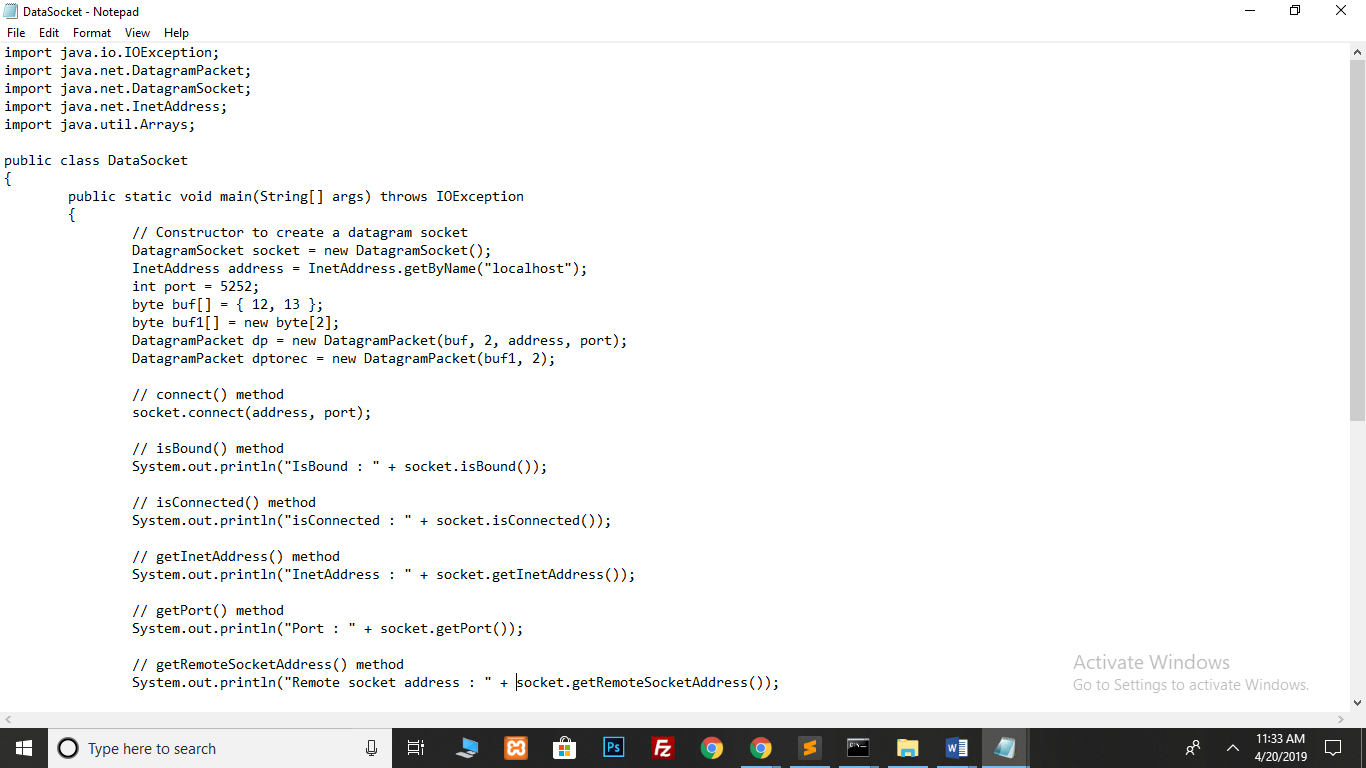


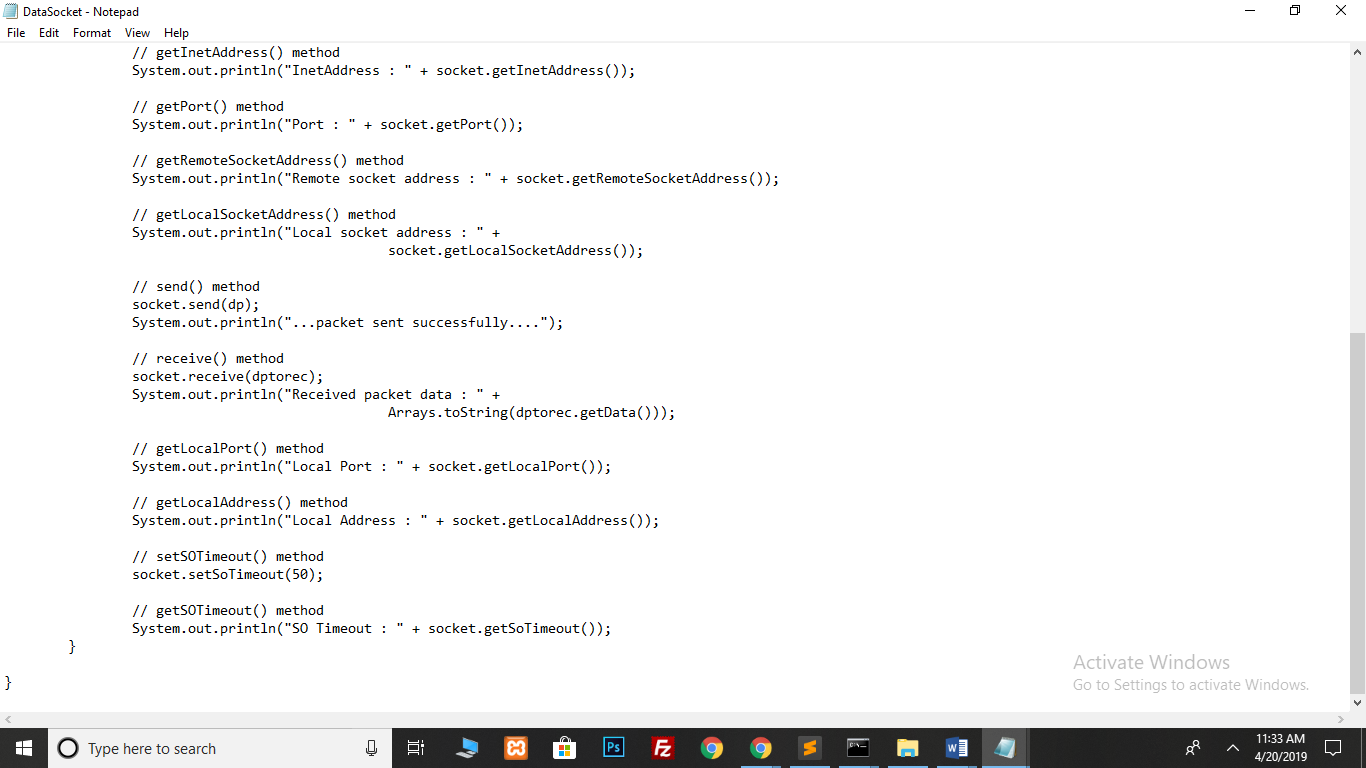


**Broadcast**

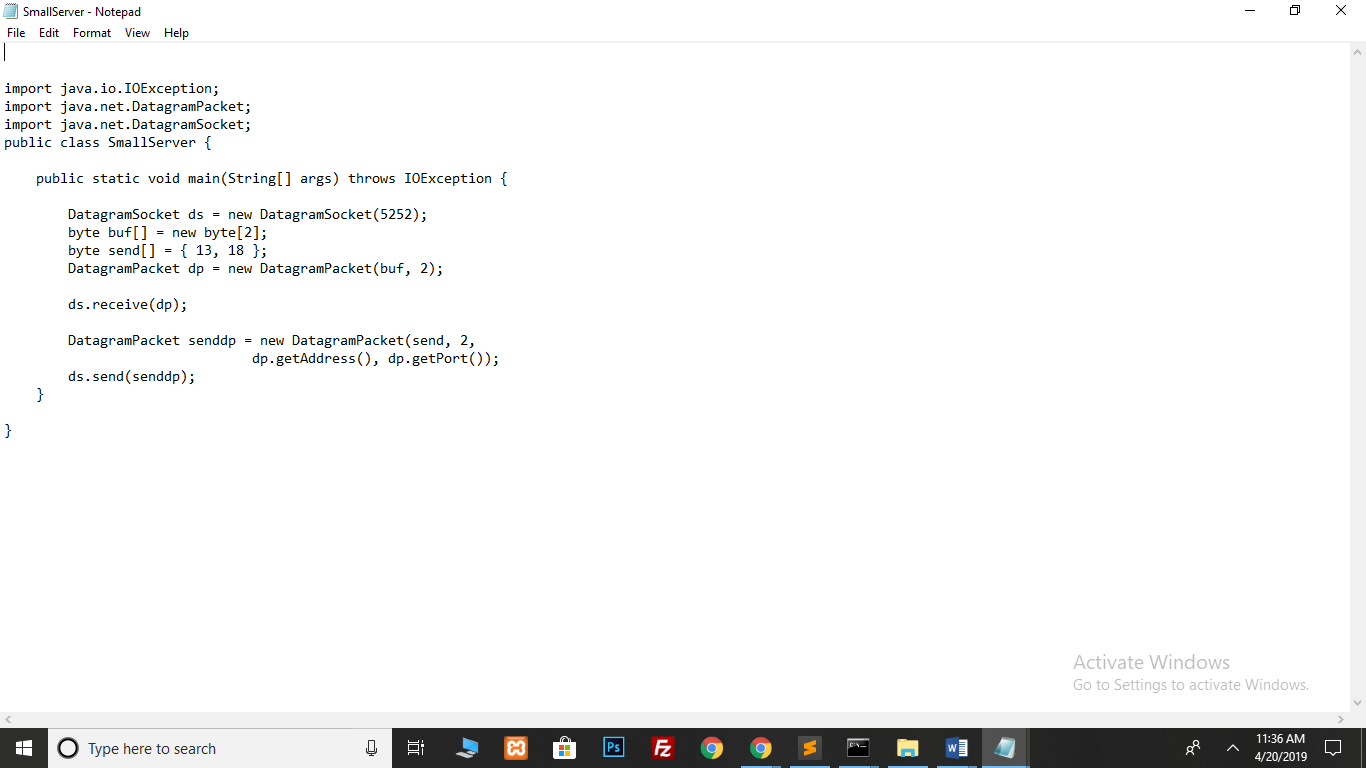
Datagram socket is a type of network socket which provides connection-less point for sending and receiving packets. Every packet sent from a datagram socket is individually routed and delivered. It can also be used for sending and receiving broadcast messages.

**Broadcaster**

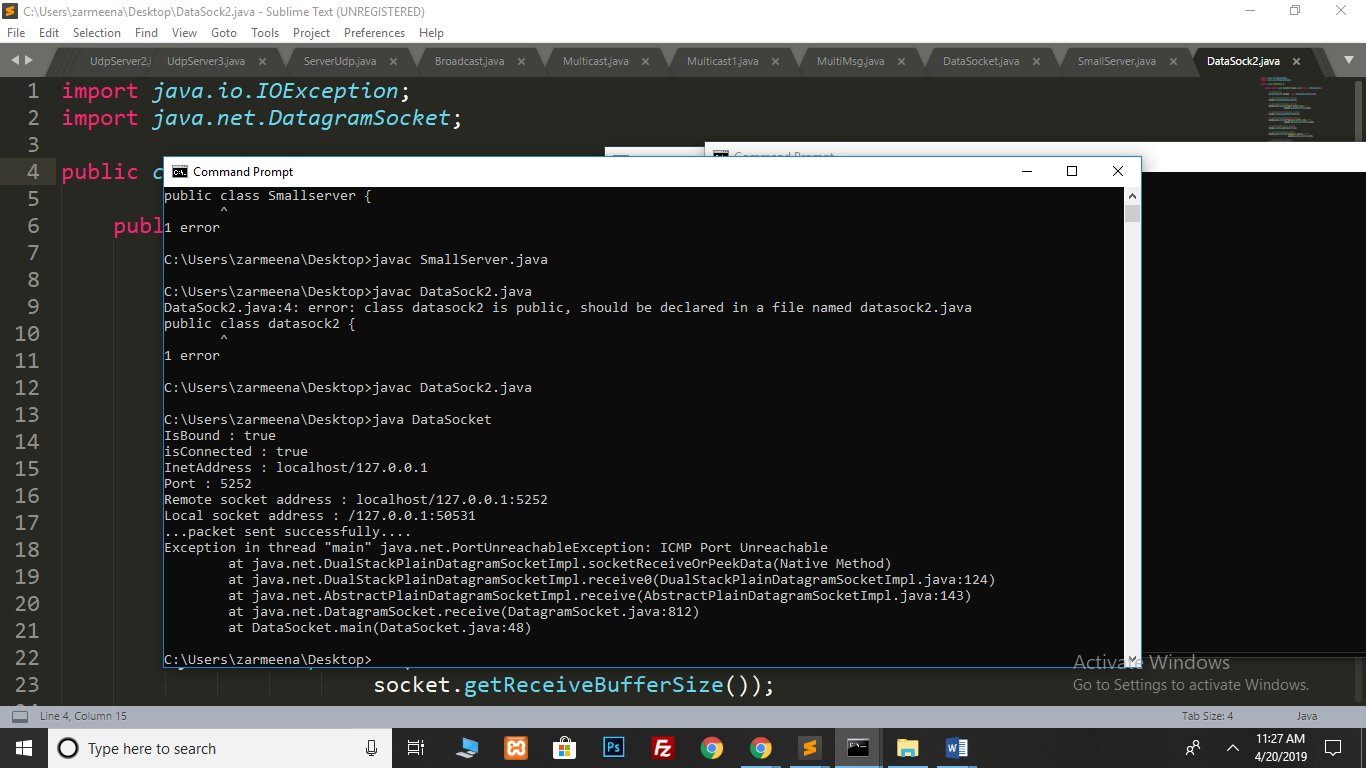




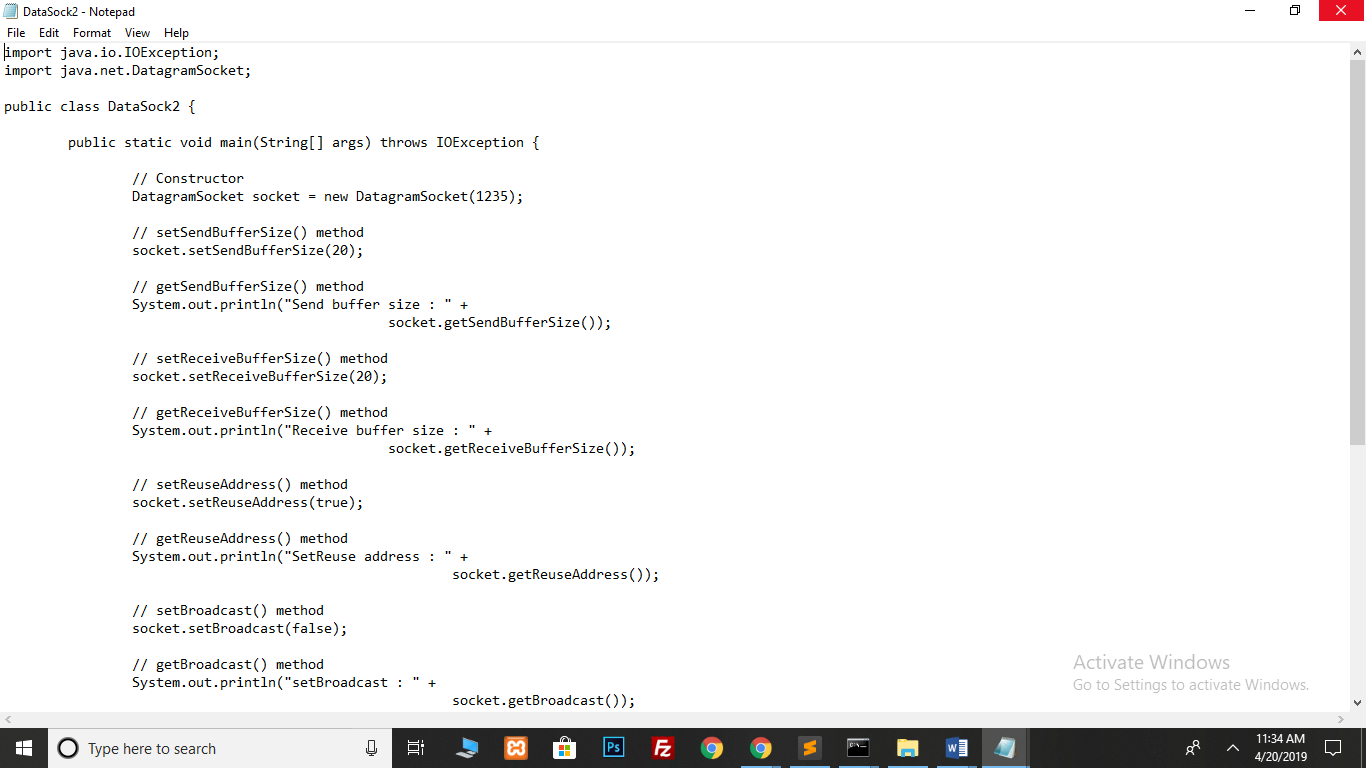
To test the above program, a small server program is required for receiving the sent packet and for implementing receive() method. Its implementation is given below.

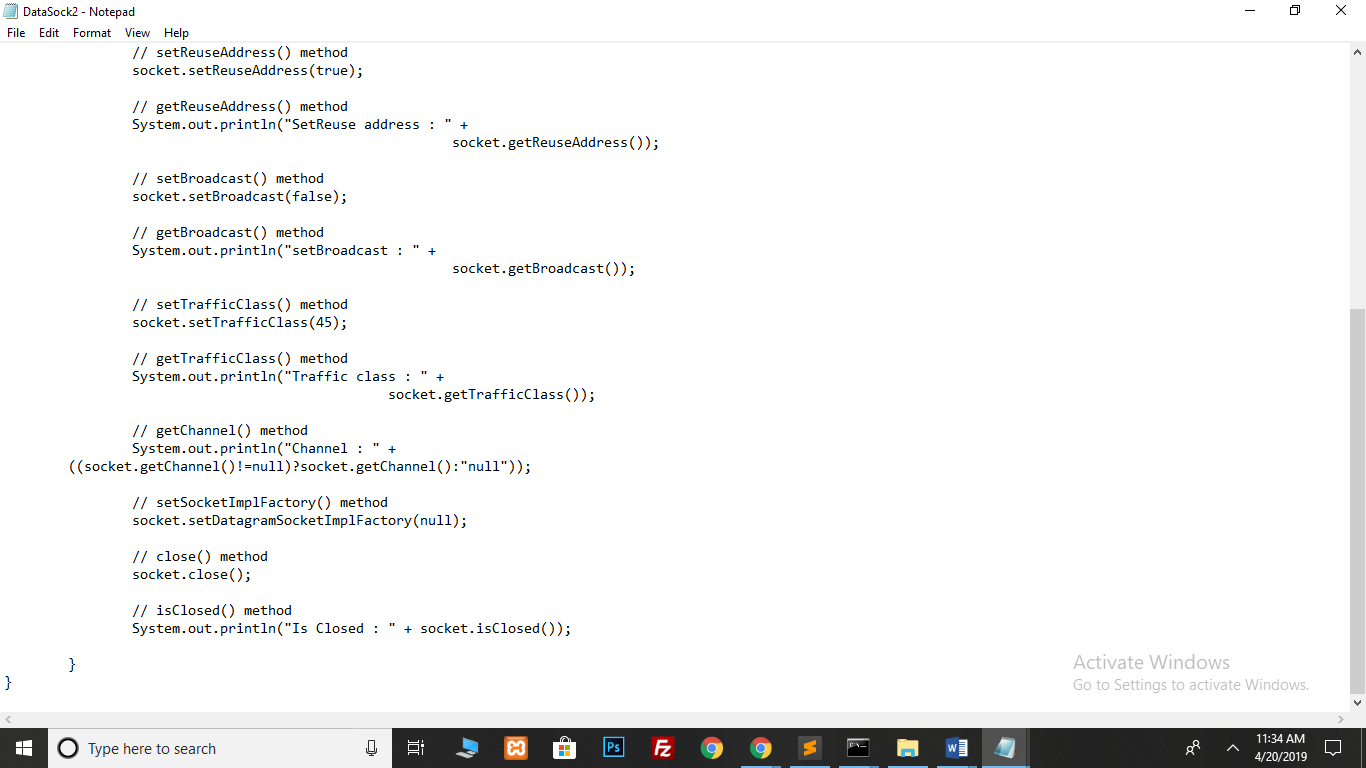


**On client side**



**Broadcast Receiver**





**Output**

