Program #5

25 points; Due Nov. 24 (Monday) at 5pm

Program Description

You will develop a simple **Internet Ping Server** running on **port 5976**, and a corresponding **Client**. The functionality provided by the Client and Server are similar to the standard ping programs available in modern operating systems, except that your Client and Server are using **UDP** rather than Internet Control Message Protocol (ICMP) to communicate with each other. The Server sits in an infinite loop listening for incoming UDP packets. When a packet comes in, the server simply sends the encapsulated data back to the client (**echoing**.) Your Client is a **Pinger** that continuously sends 10 ping messages (UDP packets) to the Server. The Client records the **RTT** (round-trip time) when the Server echoes back the UDP packet. The Client prints the minimum and maximum **RTTs,** and the average **RTT** at the end.

The Server

UDP provides applications with an unreliable transport service, because messages may get lost in the network due to router queue overflows or other reasons. Applications using UDP for communication must therefore implement any reliability they need separately in the application level (each application can implement a different policy, according to its specific needs). Your Server MUST inject artificial loss to simulate the effects of network packet loss. The Server has a parameter, **LOSS_RATE = 0.3**, which determines the percentage of packets should be lost. The Server also has another parameter, **AVERAGE_DELAY = 100 (milliseconds),** which is used to simulate transmission delay from sending a packet across the Internet. Set **AVERAGE_DELAY = 0** to find out the true **RTT** (round-trip times) of your packets. Use the following program structure.

```
import java.io.*;
import java.net.*;
import java.util.*;

public class PingServer
{
   private static final double LOSS_RATE = 0.3;
   private static final int DOUBLE = 2;
   private static final int AVERAGE_DELAY = 100; //millisecons
   private static final int PACKET_LENGTH = 1024;

   public static void main(String[] args)
   {
      try
      {
         PingServer server = new PingServer();
         server.run();
      }
      ...
   }
   //...
   public void run()
   {
      DatagramSocket serverSocket = null;
      System.out.println("Ping Server running....");
      try
      {
         Random random = new Random(new Date().getTime());
         serverSocket = new DatagramSocket(5976);
         while (true)
         {
            try
            {
               System.out.println("Waiting for UDP packet....");
               byte[] buff = new byte[PACKET_LENGTH];
               DatagramPacket packet = new DatagramPacket(buff, PACKET_LENGTH);
               //write codes for receiving the UDP packet
               //write codes for displaying the packet content on the system output
               if ( random.nextDouble() < LOSS_RATE ) //simulate packet loss
               {
                  System.out.println("Packet loss...., reply not sent.");
                  continue;
               }
               //simulate network delay
               Thread.sleep((int)(random.nextDouble() * DOUBLE * AVERAGE_DELAY));
               //write codes for making packet and sending the packet
               //write codes for displaying the packet sent message on the system output
            }
            catch (IOException e)
            {
               System.out.println("Datagram packet error...." + e);
            }
            catch (InterruptedException e)
            {
               continue;
            }
         } //end while
      }
      catch (SocketException e)
      {
         System.out.print("DatagramSocket error..." + e);
      }
      serverSocket.close();
   } //end run()
```

The Client

Your Client should send **10 ping requests** to the Server, with **one second** (1000 milliseconds) **interval**. Each message contains a payload of data that includes the keyword PING, a sequence number, and a timestamp. After sending each packet, the client waits up to one second to receive a reply. If one second goes by without a reply from the server, then the client assumes that its packet or the server's reply packet has been lost in the network. And the Client prints an error message identifying packet loss to the system output. You will need **setSoTimeout()** method in **DatagramSocket** class to set the timeout value on a datagram socket. Set **5 second timeout** (5000 milliseconds) for collecting pings at the end when all pings are sent, just to wait for possible delayed replies from the Server. When developing your code, you should run the ping server on your machine, and test your client by sending packets to localhost (or, 127.0.0.1).

(1) **Ping Message Format** (the payload)
Each message the Client sent to the Server is a String with the following format:
PING□sequence_number□timestamp

The following examples show the first 2 ping messages where the sequence number starts at 0, and timestamp is the time (use the **getTime()** method in **Date** class) when the client sends the message.
PING□0□1414956775953
PING□1□1414956776967

(2) **Required Classes**
❖ **PingPacket class** with the following private data members methods.

```
private InetAddress hostAddr;
private int port;
private String payload;

/**
* This is the constructor.
*/
public PingPacket(InetAddress hostaddr, int port, String payload)
```

```
/**
* This method returns the host IP address
*/
public InetAddress getHost()

/**
* This method returns the port number
*/
public int getPort()

/**
* This method returns the payload
*/
public String getPayload()
```

❖ **UDPPinger class** with the following methods.

```
/**
* This method creates a Datagram socket.
*/
public void createSocket()

/**
* Given a PingPacket, this method makes an UDP packet using the payload, the
* size, the destination address and the destination port, and sends the packet.
*/
public void sendPing(PingPacket pm)

/**
* This method allocates a buffer with MAX_PACKET_LENGTH, receives the response
* from the Server (socket), and displays the response on the system output.
* size, the destination address and the destination port, and sends the packet.
*/
public PingPacket receivePing() throws SocketTimeoutException
```

❖ **UDPClient class**; this is your Client program.

```
public class UDPClient extends UDPPinger implements Runnable
{
   static final int NUM_PINGS = 10;
   static final int TIMEOUT = 1000; //1 second
   static final int REPLY_TIMEOUT = 5000; //5 seconds
   ...
   static boolean[] replies = new boolean[NUM_PINGS]; //keep track of the replies
   static long[] rtt = new long[NUM_PINGS]; //keep track of the RTTs
   ...
   public UDPClient(String host, int port) //constructor
   {
      ...
   }
   public static void main(String arg[])
   {
      ...
      UDPClient client = new UDPClient(host, port);
      client.run();
   }
   @Override
   public void run()
   {
      //create the socket and set time out to 1 second.
      //set up a for loop to send 10 ping messages, and record replies and RTTs.
      //if less than 10 replies, set time out to 5 seconds, just in case replies are on the way.
      //compute and print the average RTT, minimum and maximum RTTs.
   }
}
```

```
/**
* Given the payload of a UDP packet, this helper method calculates the RTT,
* and uses the ping number as the index to store the RTT in the array.
* RTT = (current timestamp) – (previous timestamp when sending the packet)
* A counter is used to keep track of the number of valid replies from the
* Server.
*/
private void handleRTT(String payload)
```

Program Requirement

1. Your programs must not crash under any situation.
2. You must try-catch everything and print out appropriate error messages identifying the specific exception.
3. You must display information about ping messages on Server console and Client console. For example,
   - **Client console sample output**

```
Contacting host: 127.0.0.1 at port 8888
Received packet from /127.0.0.1 8888 Thu Nov 13 15:53:24 CST 2014
receivePing...java.net.SocketTimeoutException: Receive timed out
Received packet from /127.0.0.1 8888 Thu Nov 13 15:53:26 CST 2014
Received packet from /127.0.0.1 8888 Thu Nov 13 15:53:26 CST 2014
Received packet from /127.0.0.1 8888 Thu Nov 13 15:53:26 CST 2014
Received packet from /127.0.0.1 8888 Thu Nov 13 15:53:26 CST 2014
receivePing...java.net.SocketTimeoutException: Receive timed out
Received packet from /127.0.0.1 8888 Thu Nov 13 15:53:27 CST 2014
Received packet from /127.0.0.1 8888 Thu Nov 13 15:53:27 CST 2014
Received packet from /127.0.0.1 8888 Thu Nov 13 15:53:27 CST 2014
PING 0: true RTT: 212
PING 1: false RTT: 1000
PING 2: true RTT: 87
PING 3: true RTT: 125
PING 4: true RTT: 197
PING 5: true RTT: 48
PING 6: false RTT: 1000
PING 7: true RTT: 117
PING 8: true RTT: 29
PING 9: true RTT: 57
Minimum = 29ms, Maximum = 1000ms, Average = 287.2ms.
```

   - **Server console sample output**

```
Ping Server running....
Waiting for UDP packet....
Received from: /127.0.0.1 PING 0 1415915604764
Reply sent.
Waiting for UDP packet....
Received from: /127.0.0.1 PING 1 1415915604976
Packet loss...., reply not sent.
Waiting for UDP packet....
Received from: /127.0.0.1 PING 2 1415915605990
Reply sent.
Waiting for UDP packet....
```

4. You MUST follow the software development ground rules from CS2430 (posted on D2L.)
5. You have the option to work on this program with a partner from any sections. The **deadline for signing up pair programming is 11/12**. One of the group members is responsible for submitting the project folder to S:\Courses\CSSE\changl\cs3830\1dropBox by the due date.
6. You MUST DEMO your program by the due date. If you work as a pair, both of you MUST show up for demoing. Demo schedule will be announced on D2L.

Program Grading

Exceptions/Violations	**Each Offense**	**Max Off**
Program not running	25	25
Failed to ping localhost	25	25
Failed to ping a remote Server	10	10
For each method/class required but not implemented	3	10
Incorrect minimum, maximum, average RTT	3	10
Incorrect ping message format (payload)	3	3
Improper handling try/catch exceptions	1	3
Improper or insufficient messages	1	3
Not complying CS2430 programming ground rules	0.5	2