This is a closed book exam. You have 150 minutes to complete the exam. There are 7 questions in total, and the point values for the questions are shown in the table below. Your answers should fit in the space provided in the exam booklet. Paper for scrap work will be made available during the examination.

Good Luck!

<table>
<thead>
<tr>
<th>Question</th>
<th>Point</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

I have neither given nor received unauthorized aid on this examination.

NAME: ____________________________________________________

SIGNATURE: _______________________________________________
1) On the next page, you will find the text of a method inspired by research on automatic steering for automobiles done by one of our department’s alums, Dean Pomerleau. Dean was working with a group that was writing software to control the steering mechanisms of a car equipped with a digital camera that provided snapshots of the road ahead like that shown below.

Dean and his colleagues observed that the center of the lane would usually contain darker pixels than the edges of the lane. The method on the next page is designed to produce a filtered version of a “road ahead” image that might be used in identifying the center of the lane based on this observation. This method computes the average grayness of each column of pixels in the image and determines the minimum and maximum value of all these column averages. Then, it replaces all of the pixels in each column with the average gray value of the columns while scaling the gray values so that the darkest of all the columns will be black and the lightest will be white. The result of applying this method to the image above looks like this:

As you can see, the center of the lane stands out pretty clearly now.

The code for this method includes five loops. In the code, each loop is identified by a number in a comment line that precedes the loop. After the code you will find several questions about these loops.
// Replace a grayscale image with an image in which all the pixels in 
// each column have been replaced by the average gray level of the 
// column and the gray levels have been stretched to extend from 0 to 
// 255.
public SImage filter( SImage original ) {
    int picWidth = original.getWidth(  );
    int picHeight = original.getHeight(  );

    // Turn the pixels of the image into a 2D array of integers
    int [][] pixels = original.getPixelArray();

    // LOOP 1: Computes average grayness of each column
    int [] grayness = new int[ picWidth ];
    for ( int x = 0; x < picWidth; x++ ) {
        int graySum = 0;
        for ( int y = 0; y < picHeight; y++ ) {
            graySum = graySum + pixels[x][y];
        }
        grayness[x] = graySum / picHeight;
    }

    // LOOP 2: Find minimum of average graynesses
    int minGray = grayness[ 0 ];
    for ( int col = 0; col < picWidth; col++ ) {
        if ( minGray > grayness[ col ] ) {
            minGray = grayness[ col ];
        }
    }

    // LOOP 3: Find maximum of average graynesses
    int maxGray = grayness[ 0 ];
    for ( int col = 0; col < picWidth; col++ ) {
        if ( maxGray < grayness[ col ] ) {
            maxGray = grayness[ col ];
        }
    }

    // LOOP 4: Stretch average grays over 0-255 range
    for ( int col = 0; col < picWidth; col++ ) {
        grayness[col] =
        255*( grayness[col] - minGray ) / (maxGray-minGray);
    }

    // LOOP 5: Fill cols of final image with stretched gray levels
    for ( int x = 0; x < pixels.length; x++ ) {
        for ( int y = 0; y < pixels[0].length; y++ ) {
            pixels[ x ][ y ] = grayness[ x ];
        }
    }

    // Replace the image pixels with the updated array values.
    return new SImage( pixels );
}
a) Suppose that LOOP 1 had been replaced by the code below (the changed line is shown in bold face):

```java
// LOOP 1: Computes average grayness of each column
int [] grayness;
for ( int x = 0; x < picWidth; x++ ) {
    int graySum = 0;
    for ( int y = 0; y < picHeight; y++ ) {
        graySum = graySum + pixels[x][y];
    }
    grayness[x] = graySum / picHeight;
}
```

Describe exactly how the new program would behave assuming this is the only change made. Would the program still run to completion or would it encounter an error before it completed? If you believe the program would run without an error being detected, please describe how the final image produced would change (if at all). If you believe an error would occur, describe the error. You don’t have to quote the message BlueJ would provide word for word, but describe the error as precisely as you can. Rather than just saying a red error message would appear, try to specify whether the error would be a null pointer, an out or range array index, or something else. Indicate whether the error would be detected when you tried to compile the program or while the program was running. If it matters, assume that the image being processed is the one shown on the first page of the problem statement.

The revised version of the code for this loop declares grayness but fails to associate it with an actual array. Since grayness is declared as a local variable, Java will notice this error and warn the programmer that the variable might be uninitialized when the program is compiled. If grayness had been an instance variable, the error would have resulted in a NullPointerException when the program was run.

b) Suppose that LOOP 2 had been replaced by the code shown below:

```java
// LOOP 2: Find minimum of average graynesses
int minGray = grayness[ 0 ];
for ( int col = 0; col > picWidth; col++ ) {
    if ( minGray > grayness[ col ] ) {
        minGray = grayness[ col ];
    }
}
```

Describe exactly how the new program would behave assuming this is the only change made to the entire program (following the same guidelines provided for part(a)).

With the revised version of the code for this loop, the body of the loop would never be executed since col would never be greater than picWidth. As a result, minGray would be left equal to the value of grayness[ 0 ] rather than being set equal to the actual minimum value. As a result, most columns would be assigned brightness values that were smaller than desired, including negative brightness values for the columns that should be darkest.
c) Suppose that the loop 3 is replaced by the loop shown below. Describe exactly how the new program would behave assuming this is the only change made to the entire program (following the same guidelines provided for part(a)).

```c
// LOOP 3: Find maximum of average graynesses
int maxGray = minGray;
for ( int col = 0; col < picWidth; col++ ) {
    if ( maxGray < grayness[ col ] ) {
        maxGray = grayness[ col ];
    }
}
```

The revised loop would actually produce the same outcome as the original loop. The final value desired for maxGray must be greater than or equal to the correct value for minGray. As long as maxGray is initialized to any value smaller than the maximum of the values, the loop will correctly find the maximum.

d) Suppose that the last loop is replaced by the loop shown below. Describe exactly how the new program would behave assuming this is the only change made to the entire program (following the same guidelines provided for part(a)).

```c
// LOOP 5: Fill cols of final image w/ stretched gray levels
for ( int x = 0; x < pixels.length; x++ ) {
    for ( int y = 0; y < pixels[0].length; y++ ) {
        pixels[ y ][ x ] = grayness[ x ];
    }
}
```

This version confuses the rows and columns of the result image. Since the number of rows in the image is unlikely to equal the number of columns, this is likely to result in an ArrayIndexOutOfBoundsException exception.
2) Consider the switched network shown below:

In this network there are two possible paths between A and B. One path goes through the switch/router S\(_1\), the other through S\(_2\). As shown in the diagram, assume that the wires connecting S\(_1\) to A and B are of equal length, L\(_1\), and that the wires connecting S\(_2\) to A and B are both of length L\(_2\). As usual, assume that the propagation speed of signals through these wires is just \(c\), the speed of light.

In addition, assume that all messages transmitted to or from a given switch are transmitted at the same rate. Let R\(_1\) be the rate (in bits/second) at which both A and B exchange data with S\(_1\), and R\(_2\) be the rate at which A and B exchange data with S\(_2\).

a) Suppose that a single packet contains D bits. Give a formula for the time, \(P_i\), required to send such a packet from A to B through S\(_i\). Your formula should account for the time that elapses between when A begins to transmit the first bit of the packet until B receives the final bit of the packet.

\[
S_i = 2 \times \left( \frac{D}{R_i} + \frac{L_i}{c} \right)
\]

b) Given that L\(_1\) > L\(_2\) as the diagram above suggests, if R\(_1\) \(\leq\) R\(_2\), then going through S\(_2\) would always be the best route to use when sending a packet between A and B. If, however, R\(_1\) > R\(_2\) then for sufficiently large packets, it may be faster to send a packet through S\(_1\). Give a formula for the size of the smallest packet for which S\(_1\) would provide the better route.

\[
D > R_1 R_2 \times \left( \frac{L_1 - L_2}{c (R_1 - R_2)} \right)
\]
3) There are schemes for error detection and error correction that require fewer parity bits than those presented in class. In fact, one of these schemes, the Hamming code, is optimal in that it makes it possible to correct errors while using as few parity bits as theoretically possible. In the special case of sending just four data bits, there is a fairly intuitive way to understand the Hamming code.

Suppose that the four data bits of the message that need to be sent are named m1, m2, m3 and m4. We associate one parity bit with each of the following subsets of the four bits: \{m1, m2, m3\}, \{m1, m2, m4\}, and \{m2, m3, m4\}. That is, we add a total of three parity bits. We refer to these bits as p1, p2, and p3. We transmit the sequence of bits m1, m2, m3, m4, p1, p2, p3. The following Venn diagram illustrates the relationships between the three subsets and their parity bits.

When using a single parity bit, we selected either a 0 or a 1 for the parity bit so that the total number of 1s found in the combination of the parity bit and the bits it checks is even. In this Hamming code, we will select each parity bit so that the total number of 1s found in each circle of the Venn diagram is even. For example, if the message to encode is 1001 where m1 = 1, m2 = 0, m3 = 0 and m4 = 1, we would begin by placing each of these binary values in the appropriate place in the Venn diagram to obtain:

Now, we set the parity bits so that the total number of 1s in each circle is even. For example, since the top circle contains one 1, we must set p1 to 1 so that the total number of 1s will be two, an even number. The complete set of parity values for this message is shown at the top of the next page.
We then list all the message bits followed by the parity bits to obtain the actual message to transmit: 1001101.

(a) Determine the parity bits that would be associated with the message 1101 using this scheme. Show both the complete Venn diagram and the actual seven bits that would be sent after the parity bits were added.

(b) Suppose that one bit of a message sent using this scheme is damaged in transmission and the bits received are 0010110. Determine which bit of the message was damaged and show the correct, original message. Please construct the Venn diagram for the bits of this message and explain how you can identify the damaged bit.

The parity is incorrect in the lower two circles. The only bit include in both of these circles but not in the top circle is the last 0 from the four data bits, so the intended message must have been 0011110.
(c) If two bits of a message encoded in this way are damaged, it is no longer possible to correct the error. Show an example that demonstrates this fact. That is, show two possible correct messages that could be turned into the same invalid message by changing no more than two bits in either of the original messages. You may find it useful to start with the invalid message from part (b).

The two message diagrammed below are both valid. The first can be turned into the damaged message described in part (b) by changing just one bit. The second can be changed into that message by changing the first and third method bits. Therefore, if more than one bit is damaged, we cannot always tell how to correct a message.

(d) When the Hamming code is used as an error correcting code, the machine that receives a message assumes that the bits received resulted from a scenario in which at most one bit was damaged. Assuming each bit is damaged with small probability, p, independent of all other bits, what is the probability that this assumption is valid? That is, what is the probability that at most one bit is damaged? Recall that the probability of at most one bit being damaged is the probability that exactly one bit is damaged plus the probability that no bits are damaged.

The probability that no bits are damaged is \((1-p)^7\). The probability that any particular single bit is damage is \(p(1-p)^6\) so the probability that exactly one bit is damaged is \(7p(1-p)^6\). Therefore the probability that at most one bit is damaged is \((1-p)^7 + 7p(1-p)^6\).
4) Write a method which takes an array of Strings and returns the index of the longest string in the array. You may assume the the array is full of Strings.

```java
public int getIndexOfLongest( String[] strArray ) {

    int position = 0;

    for ( int i = 1; i < strArray.length; i++ ) {
        if ( strArray[position].length < strArray[i].length ) {
            position = i;
        }
    }

    return position;
}
```
5) Use Dijkstra’s Algorithm to complete the tables below using B (NOT A!) as the starting point.
- Mark starting point as **KNOWN** with length 0
- Identify each neighbor of start as **ADJACENT**
- Set first step of each neighbor of start to itself
- Set route length of each neighbor to first step distance
- While you don't know how to reach all the cities:
  - Select adjacent city with shortest route
  - Identify adjacent city with shortest route as **KNOWN**
  - Mark neighbors of new **KNOWN** city that were **DISTANT** as **ADJACENT**
  - Update path lengths and record first steps to **ADJACENT** neighbors of new **KNOWN** city.

<table>
<thead>
<tr>
<th>Cities</th>
<th>Best route length</th>
<th>First step</th>
<th>Status</th>
<th>Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>A</td>
<td><strong>Distant</strong></td>
<td>B - 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Adjacent</strong></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>-</td>
<td><strong>Known</strong></td>
<td>A - 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>C</td>
<td><strong>Distant</strong></td>
<td>B - 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Adjacent</strong></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>C</td>
<td><strong>Distant</strong></td>
<td>A - 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Adjacent</strong></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>10</td>
<td>C</td>
<td><strong>Distant</strong></td>
<td>A - 6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Adjacent</strong></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>11</td>
<td>A</td>
<td><strong>Distant</strong></td>
<td>B - 3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C</td>
<td><strong>Adjacent</strong></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>9</td>
<td>G</td>
<td><strong>Distant</strong></td>
<td>B - 5</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>H</td>
<td><strong>Adjacent</strong></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>4</td>
<td>H</td>
<td><strong>Distant</strong></td>
<td>C - 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Adjacent</strong></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>10</td>
<td>A</td>
<td><strong>Distant</strong></td>
<td>D - 1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>C</td>
<td><strong>Adjacent</strong></td>
<td></td>
</tr>
</tbody>
</table>
6) Below you will find a series of packets captured using a program similar to TCPCapture. For each packet, the information provided includes source and destination IP addresses and TCP ports, the settings of all TCP flags, the TCP sequence and acknowledgment numbers, the length (Len:) of the data in the TCP packet, and the total length of the IP packet used to transport the TCP packet (but not of the hardware packet used to transport the IP packet). The packets are numbered 1 through 22 in the trace.

Please answer the questions found after the trace.

<table>
<thead>
<tr>
<th>No.</th>
<th>Source</th>
<th>Source Port</th>
<th>Destination</th>
<th>Dest Port</th>
<th>Total length</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>60</td>
<td>Seq: 1146917693, Len: 0 Flags: 0x0002 (SYN)</td>
</tr>
<tr>
<td>2</td>
<td>137.165.8.2</td>
<td>110</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>58</td>
<td>Seq: 2044791900, Ack: 1146917694, Len: 0 Flags: 0x0012 (SYN, ACK)</td>
</tr>
<tr>
<td>3</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>60</td>
<td>Seq: 1146917694, Ack: 2044791901, Len: 0 Flags: 0x0010 (ACK)</td>
</tr>
<tr>
<td>4</td>
<td>137.165.8.2</td>
<td>110</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>90</td>
<td>Seq: 2044791901, Ack: 1146917694, Len: 36 Flags: 0x0018 (PSH, ACK) +OK POP3 2004.87 server ready</td>
</tr>
<tr>
<td>5</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>60</td>
<td>Seq: 1146917694, Ack: 2044791937, Len: 0 Flags: 0x0010 (ACK)</td>
</tr>
<tr>
<td>6</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>64</td>
<td>Seq: 1146917694, Ack: 2044791937, Len: 10 Flags: 0x0018 (PSH, ACK) USER tom</td>
</tr>
<tr>
<td>7</td>
<td>137.165.8.2</td>
<td>110</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>96</td>
<td>Seq: 2044791937, Ack: 1146917704, Len: 41 Flags: 0x0018 (PSH, ACK) +OK User name accepted</td>
</tr>
<tr>
<td>8</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>60</td>
<td>Seq: 1146917704, Ack: 2044791978, Len: 0 Flags: 0x0010 (ACK)</td>
</tr>
<tr>
<td>9</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>73 PASS cantremember</td>
<td>Seq: 2044791978, Ack: 1146917704, Len: 19 Flags: 0x0018 (PSH, ACK)</td>
</tr>
<tr>
<td>10</td>
<td>137.165.8.2</td>
<td>110</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>54</td>
<td>Seq: 2044791978, Ack: 1146917723, Len: 0 Flags: 0x0010 (ACK)</td>
</tr>
<tr>
<td>11</td>
<td>137.165.8.2</td>
<td>110</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>70</td>
<td>Seq: 2044791978, Ack: 1146917723, Len: 16 Flags: 0x0018 (PSH, ACK) ERR Bad login</td>
</tr>
<tr>
<td>12</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>54</td>
<td>Seq: 1146917723, Ack: 2044791994, Len: 0 Flags: 0x0010 (ACK)</td>
</tr>
<tr>
<td>13</td>
<td>137.165.8.2</td>
<td>110</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>70</td>
<td>Seq: 2044791994, Ack: 1146917723, Len: 16 Flags: 0x0018 (PSH, ACK) ERR Bad login</td>
</tr>
<tr>
<td>14</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>54</td>
<td>Seq: 1146917723, Ack: 2044791994, Len: 0 Flags: 0x0010 (ACK)</td>
</tr>
<tr>
<td>15</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>60 quit</td>
<td>Seq: 1146917723, Ack: 2044791994, Len: 6 Flags: 0x0018 (PSH, ACK)</td>
</tr>
<tr>
<td>16</td>
<td>137.165.8.2</td>
<td>110</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>68</td>
<td>Seq: 2044791994, Ack: 1146917729, Len: 14 Flags: 0x0018 (PSH, ACK) +OK Sayonara</td>
</tr>
<tr>
<td>17</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>54</td>
<td>Seq: 1146917729, Ack: 2044792008, Len: 0 Flags: 0x0010 (ACK)</td>
</tr>
<tr>
<td>18</td>
<td>137.165.8.2</td>
<td>110</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>54</td>
<td>Seq: 2044792008, Ack: 1146917729, Len: 0 Flags: 0x0011 (FIN, ACK)</td>
</tr>
<tr>
<td>19</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>60</td>
<td>Seq: 1146917729, Ack: 2044792008, Len: 0 Flags: 0x0010 (ACK)</td>
</tr>
<tr>
<td>20</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>60</td>
<td>Seq: 1146917729, Ack: 2044792009, Len: 0 Flags: 0x0011 (FIN, ACK)</td>
</tr>
<tr>
<td>21</td>
<td>70.109.228.34</td>
<td>49925</td>
<td>137.165.8.2</td>
<td>110</td>
<td>60</td>
<td>Seq: 1146917729, Ack: 2044792009, Len: 0 Flags: 0x0011 (FIN, ACK)</td>
</tr>
<tr>
<td>22</td>
<td>137.165.8.2</td>
<td>110</td>
<td>192.168.1.107</td>
<td>49925</td>
<td>54</td>
<td>Seq: 2044792009, Ack: 1146917750, Len: 0 Flags: 0x0010 (ACK)</td>
</tr>
</tbody>
</table>
a) What is the IP address of the server being contacted?

137.165.8.2

b) How many distinct TCP connections are involved in this sequence of packets?

Just 1

c) Which computer initiates the termination of the first TCP connection established, the client or the server? In what packet does this occur?

The server sends the first FIN in packet 18.

d) What higher level protocol is using TCP to transport data in the interaction shown?

POP

e) What is the purpose/role of packet number 8? Be as specific as possible.

It acknowledges the receipt of the data received in packet 7, a +OK from the server.

f) In case you haven’t already noticed, something interesting happens between packets 11 and 14. The password included at line 9 of the trace is (unsurprisingly) incorrect. As a result, the server sends back an error message. Oddly, two copies of this message appear in packets 11 and 13. Similarly, packets 20 and 21 appear to be identical. What is/are the likely explanation(s) for each of these two pairs of duplicated packets?

The server seems to have retransmitted the “-ERR” message seen in packet 11. It is clear that this packet was actually received by the client, because the client acknowledges packet 11 in packet 12. The only way the server could fail to be aware of this is if packet 12 was lost (or at least significantly) delayed in the network.

On the other hand, with packet 20 we have no way of knowing whether packet 20 was not delivered or whether it was delivered but its acknowledgement was lost. Since the trace only shows packets that reached the client’s network.
7) Below you will find a version of a recursive class ParticipantList that could be used to store buddies involved in a conversation in an IM client. This class uses a new type of connection called an IMConnection to send IMs. An instance variable has been added to this class to keep track of whether a user is online or not and a setOnline method has been defined so that an invocation of the form:

\[
\text{participants.setOnline( "somebuddy", false );}
\]
can be used to record the fact that the user with name “somebuddy” has gone offline, and an invocation

\[
\text{participants.setOnline( "somebuddy", true );}
\]
can be used when “somebuddy” came online again (probably due to an UPDATE_BUDDY message).

```java
public class ParticipantList {
    private boolean empty;
    private String buddyName;
    private boolean online;
    private ParticipantList rest;

    public ParticipantList() {
        empty = true;
    }

    public ParticipantList(String buddy, ParticipantList existingList) {
        empty = false;
        buddyName = buddy;
        online = true;
        rest = existingList;
    }

    public void forwardToAll( String senderName, String message, IMConnection conn ) {
        if ( !empty ) {
            rest.forwardToAll( senderName, message, conn);
            if ( !buddyName.equals( senderName ) ) {
                conn.sendIM( buddyName,
                           "From " + senderName + ": " + message );
            }
        }
    }

    public void setOnline( String name, boolean isOnline ) {
        if ( !empty ) {
            if ( buddyName.equals( name ) ) {
                online = isOnline;
            } else {
                rest.setOnline( name, isOnline );
            }
        }
    }
}
```
a) Although the class definition above makes it possible to keep track of whether each buddy in a participant list is online or not, it doesn’t use this information in any way. One obvious way it could be used is to modify the `forwardToAll` method so that messages are only sent to the buddies in the list who are currently online. Three attempts to define such a modified version of `forwardToAll` are shown below. Indicate which version is correct. Briefly explain what is wrong with the versions you rejected.

i. ```java
   public void forwardToAll( String senderName,
                              String message, IMConnection conn) {
      if ( !empty && online ) {
         rest.forwardToAll( senderName, message, conn );
         if ( !buddyName.equals( senderName ) ) {
            conn.sendIM( buddyName,
                         "From " + senderName + ": " + message );
         }
      }
   }
```

ii. ```java
    public void forwardToAll( String senderName,
                              String message, IMConnection conn) {
       if ( !empty ) {
          if ( !buddyName.equals( senderName ) && online ) {
             conn.sendIM( buddyName,
                          "From " + senderName + ": " + message );
          }
          rest.forwardToAll( senderName, message, conn );
       }
    }
```

iii. ```java
    public void forwardToAll( String senderName,
                              String message, IMConnection conn) {
       if ( online && !buddyName.equals( senderName ) ) {
          conn.sendIM( buddyName,
                      "From " + senderName + ": " + message );
       }
       if ( !empty ) {
          rest.forwardToAll( senderName, message, conn );
       }
    }
```
b) Another use of the instance variable is in a method named `isOnline` that indicates whether a particular user is online or not. That is, the invocation:

\[ \text{participants.isOnline( "somebuddy" )} \]

returns true if the name “somebuddy” appears in the `participants` list within an object for which the variable `online` is true. The method returns false otherwise.

```java
public boolean isOnline( String name ) { 
    if ( empty ) { 
        return false; 
    } else if ( name.equals( buddyName ) ) { 
        return online; 
    } else { 
        return rest.isOnline( name ); 
    }
}
```