1 Objectives

(i.) Learn to work with acknowledgements.

(ii.) Understand the necessity of acknowledgements in networking.

2 Tools

(i.) Java Socket Programming

(ii.) Sublime Text

3 Procedure

3.1 Server-side

i. Defining the socket to listen to a specific port.

```
ServerSocket        SerSock         = new ServerSocket(1234);
```

ii. Specifying the port to listen for clients. Whenever a client is connected, a message is shown.

```
Socket              Sock            = SerSock.accept();
System.out.println("Client␣Connected.");
```

iii. Recieving the object from client through *ObjectInputStream*.

```
   RecieveFromClient = (DataPacket)ReadStream.readObject();
```

iv. Checking the parity of the recieved object and matching it with the parity of the object. If there is no error, the message from the object is shown. Otherwise, an error message is displayed.

```
   if(RecieveFromClient.CheckParity() == RecieveFromClient.getParity()){
       System.out.println("Message␣recieved␣correctly.");
       System.out.println(RecieveFromClient.getMessage());
     }
     else{
       System.out.println("Message␣is␣broken.");
     }
```

v. Creating another packet in response to client's object, with the header of the object that was received; and sending that to client using *ObjectOutputStream*.

```
SendToClient.CreatePacket(RecieveFromClient.getHeader());
WriteStream.writeObject(SendToClient);
```

vi. Clearing the objects so that they can reallocated later.

```
SendToClient = null;
RecieveFromClient = null;
```

vii. The connection has been terminated. This part of code releases the resources occupied *i.e. the port*.

```
SerSock.close();
Sock.close();
```

3.2 Client-side

i. Specifying the client socket to search for server in a defined port. Once a server is found and connected to it, a confirmation message is shown to the user console.

```
Socket        CliSock       = new Socket("localhost", 1234);
System.out.println("Connected␣to␣Server.");
```

ii. Creating a packet including *Header, Message* and *Protocol ID*; and sending the object to the server through *ObjectOutputStream*.

```
SendToServer.CreatePacket(iter, UserInput.nextLine(), iter*100);
WriteStream.writeObject(SendToServer);
```

iii. Recieving the object sent from server through *ObjectInputStream*.

```
RecievedFromServer = (DataPacket)ReadStream.readObject();
```

iv. Checking the parity of the object sent from the server and matching it with the parity set in the object. If there is no error, then checking for the acknowledgement. If *acknowledgement* matches the *header* of the sent object, the client is prompted with a success. Otherwise, an error message is displayed.

```
if(RecievedFromServer.CheckParity() == RecievedFromServer.getParity()){
  System.out.println("Acknowledgement␣recieved␣correctly.");
  if(RecievedFromServer.getAcknowledgement() == SendToServer.getHeader()){
    System.out.println("Server␣recieved␣correct␣frame.");
  }
  else{
    System.out.println("Server␣recieved␣wrong␣frame.");
  }
}
else{
  System.out.println("Message␣is␣broken.");
}
```

v. Clearing the objects so that they can reallocated later.

```
SendToServer = null;
RecievedFromServer = null;
```

vi. The connection has been terminated. This part of code releases the resources occupied *i.e. the port*.

```
UserInput.close();
CliSock.close();
```

3.3 Custom Class (DataPacket)

i. Defining the constructor to use in initializing the packets.

```
DataPacket(){
  setHeader(0);
  setMessage("");
  setProtocolID(0);
  setAcknowledgement(0);
  setParity();
}
```

ii. Setting the *Header, Message, ProtocolID, Acknowledgement* and *Parity* of the data packet.

```
public void   setHeader(int Header){this.Header = Header;}
public void   setMessage(String Message){this.Message = Message;}
public void   setProtocolID(int ProtocolID){this.ProtocolID = ProtocolID;}
public void setParity(){
  this.Parity = true;

  if(CalculateParity(this.Header))  this.Parity = !this.Parity;
  if(CalculateParity(this.Message)) this.Parity = !this.Parity;
  if(CalculateParity(this.ProtocolID))  this.Parity = !this.Parity;
  if(CalculateParity(this.Acknowledgement)) this.Parity = !this.Parity;
}
public void   setAcknowledgement(int Header)  {this.Acknowledgement = Header;}
```

iii. Retrieving the *Header, Message, Acknowledgement* and *Parity* for the calling environment.

```
public int    getHeader(){return this.Header;}
public String   getMessage()  {return this.Message;}
public boolean  getParity(){return this.Parity;}
public int    getAcknowledgement(){return this.Acknowledgement;}
```

iv. To check parity of the object to match the data in the object.

```
public boolean CheckParity(){
  boolean parity = true;

  if(CalculateParity(this.Header)) parity = !parity;
  if(CalculateParity(this.Message)) parity = !parity;
  if(CalculateParity(this.ProtocolID))  parity = !parity;
  if(CalculateParity(this.Acknowledgement)) parity = !parity;

  return parity;
}
```

v. This method calculates the parity of the passed integer.

```
public boolean CalculateParity(int n){
  boolean parity = true;
  while(n != 0){
        parity = !parity;
        n = n & (n-1);
  }

  return parity;
}
```

vi. This method calculates the parity of the passed string by converting it to a *byte-array*.

```
public boolean CalculateParity(String str){
  boolean parity = true;
  byte[] byteArray = str.getBytes();
  int iter;

  for(iter=0; iter<byteArray.length; iter++){
    if(byteArray[iter] == 1)  parity = !parity;
  }

  return parity;
}
```

vii. Client invokes this method to create a packet that will be sent to the server.

```
public void CreatePacket(int Header, String Message, int ProtoclID){
  setHeader(Header);
  setMessage(Message);
  setProtocolID(ProtocolID);
  setAcknowledgement(0);
  setParity();
}
```

viii. Server invokes this method to create a packet that will be sent to the client.

```
public void CreatePacket(int Acknowledgement){
  setHeader(0);
  setMessage("");
  setProtocolID(0);
  setAcknowledgement(Acknowledgement);
  setParity();
}
```

4 Conclusion

We have understood the necessity of using acknowledgements in networking and the problems that can occur if it is not used or used improperly. We have also gethered knowledge about the case when responce is delayed too much.

5 Appendix

5.1 Server code

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


public class server{
  public static void main(String[] args) throws IOException, ClassNotFoundException{
    ServerSocket SerSock = new ServerSocket(1234);
    Socket Sock = SerSock.accept();
    System.out.println("Client␣Connected.");

    ObjectOutputStream WriteStream = new ObjectOutputStream(Sock.getOutputStream());
    ObjectInputStream ReadStream = new ObjectInputStream(Sock.getInputStream());
    int iter = 0;

    while(iter<10){
      DataPacket SendToClient = new DataPacket();
      DataPacket RecieveFromClient = new DataPacket();

      RecieveFromClient = (DataPacket)ReadStream.readObject();
      if(RecieveFromClient.CheckParity() == RecieveFromClient.getParity()){
        System.out.println("Message␣recieved␣correctly.");
        System.out.println(RecieveFromClient.getMessage());
      }
      else{
        System.out.println("Message␣is␣broken.");
      }

      SendToClient.CreatePacket(RecieveFromClient.getHeader());
      WriteStream.writeObject(SendToClient);

      SendToClient = null;
      RecieveFromClient = null;
      iter++;
    }


    System.out.println("Connection␣Terminated");

    SerSock.close();
    Sock.close();
  }
}
```

5.2 Client Code

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


public class client{
  public static void main(String[] args) throws IOException, UnknownHostException,
      ClassNotFoundException{
    Socket CliSock = new Socket("localhost", 1234);
    System.out.println("Connected␣to␣Server.");

    ObjectOutputStream WriteStream = new ObjectOutputStream(CliSock.getOutputStream());
    ObjectInputStream ReadStream = new ObjectInputStream(CliSock.getInputStream());
    Scanner UserInput = new Scanner(System.in);
    int iter = 0;

    while(iter<10){
      DataPacket SendToServer = new DataPacket();
      DataPacket RecievedFromServer = new DataPacket();

      SendToServer.CreatePacket(iter, UserInput.nextLine(), iter*100);

      WriteStream.writeObject(SendToServer);
      RecievedFromServer = (DataPacket)ReadStream.readObject();
```

```
24
25        if(RecievedFromServer.CheckParity() == RecievedFromServer.getParity()){
26          System.out.println("Acknowledgement recieved correctly.");
27          if(RecievedFromServer.getAcknowledgement() == SendToServer.getHeader()){
28            System.out.println("Server recieved correct frame.");
29          }
30          else{
31            System.out.println("Server recieved wrong frame.");
32          }
33        }
34        else{
35          System.out.println("Message is broken.");
36        }
37
38        SendToServer = null;
39        RecievedFromServer = null;
40        iter++;
41      }
42
43      UserInput.close();
44      CliSock.close();
45    }
46  }
```

5.3 Custom Class (DataPacket) Code

```
1  import java.io.Serializable;
2
3  public class DataPacket implements Serializable{
4    private static final long serialVersionUID = 123456L;
5    // members
6    private int     Header;
7    private String  Message;
8    private int     ProtocolID;
9    private boolean   Parity;
10   private int     Acknowledgement;
11
12   // methods
13   // constructor
14   DataPacket(){
15     setHeader(0);
16     setMessage("");
17     setProtocolID(0);
18     setAcknowledgement(0);
19     setParity();
20   }
21
22   // setters and getters
23   public void   setHeader(int Header) {this.Header = Header;}
24   public int getHeader() {return this.Header;}
25
26   public void   setMessage(String Message) {this.Message = Message;}
27   public String getMessage()  {return this.Message;}
28
29   public void   setProtocolID(int ProtocolID) {this.ProtocolID = ProtocolID;}
30
31   public void setParity(){
32     this.Parity = true;
33
34     if(CalculateParity(this.Header))  this.Parity = !this.Parity;
35     if(CalculateParity(this.Message)) this.Parity = !this.Parity;
36     if(CalculateParity(this.ProtocolID))  this.Parity = !this.Parity;
37     if(CalculateParity(this.Acknowledgement)) this.Parity = !this.Parity;
38   }
39   public boolean getParity() {return this.Parity;}
40
41   public void   setAcknowledgement(int Header) {this.Acknowledgement = Header;}
42   public int getAcknowledgement() {return this.Acknowledgement;}
43
44   // check the received message
45   public boolean CheckParity(){
46     boolean parity = true;
47
48     if(CalculateParity(this.Header))  parity = !parity;
```

```
49      if(CalculateParity(this.Message)) parity = !parity;
50      if(CalculateParity(this.ProtocolID))  parity = !parity;
51      if(CalculateParity(this.Acknowledgement)) parity = !parity;
52
53      return parity;
54    }
55
56    //parity checking methods
57    public boolean CalculateParity(int n){
58      boolean parity = true;
59      while(n != 0){
60            parity = !parity;
61            n = n & (n-1);
62      }
63
64      return parity;
65    }
66
67    public boolean CalculateParity(String str){
68      boolean parity = true;
69      byte[] byteArray = str.getBytes();
70      int iter;
71
72      for(iter=0; iter<byteArray.length; iter++){
73        if(byteArray[iter] == 1)  parity = !parity;
74      }
75
76      return parity;
77    }
78
79    // Methods to create the packets
80    public void CreatePacket(int Header, String Message, int ProtoclID){
81      setHeader(Header);
82      setMessage(Message);
83      setProtocolID(ProtocolID);
84      setAcknowledgement(0);
85      setParity();
86    }
87
88    public void CreatePacket(int Acknowledgement){
89      setHeader(0);
90      setMessage("");
91      setProtocolID(0);
92      setAcknowledgement(Acknowledgement);
93      setParity();
94    }
95  }
```