1 Objectives

(i.) Learn to work with data packets in networking.

(ii.) Learn the basics of error detection of data packets using parity checking.

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.

```
   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. Sending response to the client machine and terminating the connection.

```
WriteStream.writeObject(SendToClient);
System.out.println("Connection␣Terminated");
```

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

```
UserInput.close();
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. Sending the object to the server through *ObjectOutputStream*.

```
WriteStream.writeObject(SendToServer);
```

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

```
RecieveFromServer = (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, the message is shown. Otherwise, an error message is displayed.

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

v. 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);
  setParity();
}
```

ii. Setting the *Header, Message, ProtocolID* 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;
}
```

iii. Retrieving the *Header, Message* and *Parity* for the caller.

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

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;

  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;
  }
```

4 Conclusion

We were able to pass objects through the socket successfully. Since the delay between sending the frame and receiving it is ignorable, there was no error induced. Due to the errorless result, we were not able to observe and handle the situations where errors can be present.

5 Appendix

5.1 Server code

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


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());
    Scanner   UserInput = new Scanner(System.in);
    DataPacket SendToClient = new DataPacket();
    DataPacket RecieveFromClient = new DataPacket();
    String toEscape;

    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.");
    }

    System.out.print("Enter␣header:␣");
    SendToClient.setHeader(UserInput.nextInt());

    toEscape = UserInput.nextLine();
    System.out.print("Enter␣message:␣");
    SendToClient.setMessage(UserInput.nextLine());

    System.out.print("Enter␣Protocol␣ID:␣");
    SendToClient.setProtocolID(UserInput.nextInt());

    SendToClient.setParity();

    WriteStream.writeObject(SendToClient);
    System.out.println("Connection␣Terminated");

    UserInput.close();
    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);
    DataPacket SendToServer = new DataPacket();
    DataPacket RecieveFromServer = new DataPacket();
    String toEscape;

    System.out.print("Enter header: ");
    SendToServer.setHeader(UserInput.nextInt());

    toEscape = UserInput.nextLine();
    System.out.print("Enter message: ");
    SendToServer.setMessage(UserInput.nextLine());

    System.out.print("Enter Protocol ID: ");
    SendToServer.setProtocolID(UserInput.nextInt());

    SendToServer.setParity();

    WriteStream.writeObject(SendToServer);
    RecieveFromServer = (DataPacket)ReadStream.readObject();

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

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

5.3 Custom Class (DataPacket) Code

```
import java.io.Serializable;

public class DataPacket implements Serializable{
  private static final long serialVersionUID = 123456L;
  // members
  private int Header;
  private String Message;
  private int   ProtocolID;
  private boolean   Parity;

  // methods
  // constructor
  DataPacket(){
    setHeader(0);
    setMessage("");
    setProtocolID(0);
    setParity();
  }

  // setters and getters
  public void   setHeader(int Header){this.Header = Header;}
  public int getHeader(){return this.Header;}

  public void   setMessage(String Message){this.Message = Message;}
```

```
25    public String   getMessage()  {return this.Message;}
26
27    public void   setProtocolID(int ProtocolID){this.ProtocolID = ProtocolID;}
28
29    public void setParity(){
30      this.Parity = true;
31
32      if(CalculateParity(this.Header))  this.Parity = !this.Parity;
33      if(CalculateParity(this.Message)) this.Parity = !this.Parity;
34      if(CalculateParity(this.ProtocolID))  this.Parity = !this.Parity;
35    }
36    public boolean  getParity() {return this.Parity;}
37
38    // check the received message
39    public boolean CheckParity(){
40      boolean parity = true;
41
42      if(CalculateParity(this.Header))  parity = !parity;
43      if(CalculateParity(this.Message)) parity = !parity;
44      if(CalculateParity(this.ProtocolID))  parity = !parity;
45
46      return parity;
47    }
48
49    //parity checking functions
50    public boolean CalculateParity(int n){
51      boolean parity = true;
52      while(n != 0){
53            parity = !parity;
54            n = n & (n-1);
55      }
56
57      return parity;
58    }
59
60    public boolean CalculateParity(String str){
61      boolean parity = true;
62      byte[] byteArray = str.getBytes();
63      int iter;
64
65      for(iter=0; iter<byteArray.length; iter++){
66        if(byteArray[iter] == 1)  parity = !parity;
67      }
68
69      return parity;
70    }
71  }
```