Kristianstad University
SE-291 88 Kristianstad
Sweden
+46 44 250 30 00
www.hkr.se

Laboration 1

DA330

Software Engineering

15 hp (VT23)

Content

Lab 1 preparation – A distributed communication example 3
Task 1 4
EchoServer Main 4
EchoServer 4
EchoClient Main 5
EchoClient 5
Task 2 6
MultiClientServer Main 6
MultiClientServer 6
Server 6
Task 3 8
Task 4 9
Shared class: PersistentTime 9
FlattenTime 9
InflateTime 10
Lab 1 task description - Another distributed communication example 11

Lab 1 preparation – A distributed communication example

Since much of the course project will be focused on distributed communication, we might as well as soon as possible approach this subject. The contents of this preparation should be familiar to course participants, through previous courses concerning real time systems, networking, and communication. So, in the context of the current course this lab may serve as valuable repetitions. This document shows you how this can be done in IntelliJ but you may choose to use other development environment. Therefore, please note that your chosen development environment uses other ways to approach that.

This preparation is divided up into four sub tasks, and relates to simple client-server applications where the server simply echoes what is delivered from the client, through an established TCP connection. Task 1 and 2 are run locally while task 3 should run between different computers. For task 4 you may choose by yourself. For instance, you may start run locally, and when you are convinced about your program's correctness you may try to run towards another computer.

For tasks 1 and 2, the code is attached to task description. Read the code to make sure you understand it. Develop the corresponding required IntelliJ projects. Write the code from the text, and then run the programs, and make sure they work. For task 3 make the changes required to run non-locally. Task 4 is about communicating over serializable objects and may introduce further challenges.

This lab can be done individually or 2 students in a group.

Good luck!

Task 1

Read, write, and test the *EchoServer – EchoClient* example below.

EchoServer Main

```
public class Main {

    public static void main(String[] args) {

    EchoServer echoServer=new EchoServer();
    echoServer.establish();
    }

}
```

EchoServer

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

public class EchoServer {
    public EchoServer() {

    }
    public void establish() {
        ServerSocket serverSocket = null;
        try {
            serverSocket= new ServerSocket(1234);
        }catch (IOException e) {
            System.out.println("Could not listen on port: 1234");
            System.exit(-1);
        }
        Socket clientSocket = null;
        try {
             clientSocket = serverSocket.accept();
        }catch (IOException e) {
            System.out.println("Accept failed: 1234");
            System.exit(-1);
        }
        PrintWriter out=null;
        BufferedReader in = null;
        try {
                out = new PrintWriter(
                      clientSocket.getOutputStream(), true);
                in = new BufferedReader(
                        new InputStreamReader(
                            clientSocket.getInputStream()));
        }catch (IOException ioe) {
          System.out.println("Failed in creating streams");
            System.exit(-1);
        }
        String inputLine, outputLine;
        try {
            while ((inputLine = in.readLine()) != null) {
                out.println(inputLine);
                if (inputLine.equals("Bye."))
                    break;
            }
        }catch (IOException ioe) {
          System.out.println("Failed in reading, writing");
            System.exit(-1);
        }
        try {
```

```
            clientSocket.close();
            serverSocket.close();
          }catch (IOException e) {
            System.out.println("Could not close");
            System.exit(-1);
        }}}
```

EchoClient Main

```
public class Main {

    public  static  void  main(String[]  args)  {
        EchoClient echoClient = new EchoClient();
        echoClient.establish();
    }}
```

EchoClient

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

public class EchoClient {
    public EchoClient() {
    }
    public void establish() {
        Socket echoSocket = null;
        PrintWriter out = null;
        BufferedReader in = null;
        try {
            echoSocket = new Socket(InetAddress.getLocalHost(), 1234);
            out = new PrintWriter(echoSocket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(
                                        echoSocket.getInputStream()));
        } catch (UnknownHostException e) {
            System.err.println("Don't know about host.");
            System.exit(1);
        } catch (IOException e) {
            System.err.println("Couldn't get I/O");
            System.exit(1);
        }
           BufferedReader stdIn = new BufferedReader( new
           InputStreamReader(System.in));

           String userInput;

        try{
          while ((userInput = stdIn.readLine()) != null) {
                out.println(userInput);
                   if (userInput.equals("Bye."))
                      break;
                  System.out.println("echo: " + in.readLine());
                  }
                      out.close();
                      in.close();
                      stdIn.close();
                      echoSocket.close();
        } catch (IOException ioe) {
            System.out.println("Failed");
            System.exit(-1);
      }}}
```

Task 2

Extend *EchoServer* to make it able to accept several clients. You may use the same client as before. In this case as you may see below the servers have to be threaded.

MultiClientServer Main

```
public class Main {

    public static void main(String[] args) {
    MultiClientServer multiClientServer = new MultiClientServer();
    multiClientServer.start();
    }

}
```

MultiClientServer

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

public class MultiClientServer extends Thread
    { public MultiClientServer() {
    }
    public void run() {
       ServerSocket serverSocket = null;
        try {
            serverSocket= new ServerSocket(2345);
        }catch (IOException e) {
          System.out.println("Could not listen on port: 2345");
            System.exit(-1);
        }

       Socket clientSocket = null;
        while (true) {
            try {
            clientSocket = serverSocket.accept();
            Server server = new Server(clientSocket);

              server.start();
          } catch (IOException e) {
              System.out.println("Accept failed:2345");
            System.exit(-1);
            }
        }}}
```

Server

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

public class Server extends Thread {
     Socket clientSocket=null;
     public Server(Socket clientSocket) {
         this.clientSocket=clientSocket;
```

```
    }
    public void run() {
        PrintWriter out=null;
        BufferedReader in = null;
        try {
          out = new PrintWriter( clientSocket.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(
                                        clientSocket.getInputStream()));
        } catch (IOException ioe) {
            System.out.println("Failed in creating streams");
            System.exit(-1);
        }
        String inputLine, outputLine;
        try {
            while ((inputLine = in.readLine()) != null) {
                out.println(inputLine);
                System.out.println(inputLine);
                if (inputLine.equals("Bye."))
                    break;
            }
        } catch (IOException ioe) {
            System.out.println("Failed in reading, writing");
            System.exit(-1);
        }
        try {
            clientSocket.close();
            out.close();
            in.close();
        } catch (IOException ioe) {
            System.out.println("Failed in closing down");
            System.exit(-1);
        }
    }

}
```

Task 3

Extend *EchoClient* so this may establish contact with server that may reside at another computer than your own. To be able to do that you need to know the IP address of that server. Experiment with the methods below to find out about this. For instance you may use the following expression: System.out.println(serverSocket.getInetAddres().getHostAddress());

- ServerSocket
  - InetAddress getInetAddress()
    Returns the local address of this server socket.
  - int getLocalPort()
    Returns the port number on which this socket is listening
- Socket
  - InetAddress getLocalAddress()
    Gets the local address to which the socket is bound.
  - int getLocalPort()
    Returns the local port number to which this socket is bound.
  - InetAddress getInetAddress()
    Returns the address to which the socket is connected.
  - int getPort()
    Returns the remote port number to which this socket is connected.
- InetAddress
  - String getCanonicalHostName()
    Gets the fully qualified domain name for this IP address.
  - String getHostAddress()
    Returns the IP address string in textual presentation.
  - String getHostName()
    Gets the host name for this IP address.
- Socket
  - Socket(InetAddress address, int port)
    Creates a stream socket and connects it to the specified port number at the specified IP address.
  - Socket(String host, int port)
    Creates a stream socket and connects it to the specified port number on the named host.

You are welcome to experiment further with the methods above in order to increase your knowledge and experience about those.

Task 4

At https://docs.oracle.com/cd/E29542_01/apirefs.1111/e13948/org/apache/openjpa/util/Serialization.html there is an example where a time object of class PersistentTime is written to a file and then read back. Thereafter the time data that were set when the object was first created is printed on standard output.
Note that it is a java object that is written and read to and from the file.

The example consists of two programs FlattenTime, creating and writing the object to a file, and InflateTime, reading the object from the file, and printing the time on standard output.
Moreover, the two programs share a common class PersistentTime used to instantiate to hold time data, and then communicated between the two programs.

Read, write and run the code below!

Shared class: PersistentTime

```
import java.io.Serializable;
import java.util.Date;
import java.util.Calendar;

public class PersistentTime implements Serializable {
   private Date time;
   public PersistentTime() {
      time = Calendar.getInstance().getTime();
   }
   public Date getTime() {
      return time;
   }}
```

FlattenTime

```
import java.io.ObjectOutputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class FlattenTime {
   public static void main(String [] args){
   String filename = "time.ser";
   if(args.length > 0){
        filename = args[0];
      }
      PersistentTime time = new PersistentTime();
      FileOutputStream fos = null;
      ObjectOutputStream out = null;
      try {
        fos = new FileOutputStream(filename);
        out = new ObjectOutputStream(fos);
        out.writeObject(time);
        out.close();
      } catch(IOException ex){
        ex.printStackTrace();
      }
    }}
```

InflateTime

```
import java.io.ObjectInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Calendar;

public class InflateTime {
   public static void main(String [] args){
   String filename = "time.ser";
      if(args.length > 0) {
   filename = args[0];
      }
      PersistentTime time = null;
      FileInputStream fis = null;
      ObjectInputStream in = null;
      try {
        fis = new FileInputStream(filename);
        in = new ObjectInputStream(fis);
        time = (PersistentTime)in.readObject();
        in.close();
      } catch(IOException ex){
        ex.printStackTrace();
      } catch(ClassNotFoundException ex){
        ex.printStackTrace();
      }
      // print out restored time
      System.out.println("Flattened time: " + time.getTime());
      System.out.println();
      // print out the current time
      System.out.println("Currenttime:"+Calendar.getInstance().getTime());
  }
}
```

The example above uses an object of a common class to communicate between to programs. Communication is done over a file. Now return to *Task 1*, and change that example with inspiration from *Task 4*, to use a serialized object for communication instead of pure strings. For instance, for experimentation reasons let the client send a time object to the server, where after the server prints the time at the standard output.

Lab 1 task description - Another distributed communication example

Lab 1 will proceed with exercises started already in 'Preparation for Lab1'. On one hand we will open up for communication between different programs, on the other hand we will communicate objects over object streams. In this lab the communicated objects will consist of the information for designing the graphical objects, more precisely a description of JavaFX component object that is set up at one program, sent to another program and loaded in that program's Scene. Cool, isn't it! Besides from this the lab will deal with UML modelling.

Task 1
Similarly to 'Preparation for Lab1', one program, the Sender, will create a serializable object, description of the graphical object, and write it to a file. From another program, the Receiver, the object will be read and showed on that program's Scene.
To pass task 1:

- Explain the code and run the programs
- Show related UML models (class diagrams)
- Upload the code on Canvas

Task 2
Similarly to 'Preparation for Lab1', we will extend the example to communicate over a local TCP connection. That is, we run the same example as in Task 1, but instead the two programs will communicate over sockets.
To pass task 2:

- Explain the code and run the programs
- Show related UML models (class diagrams)
- In opposite to Task 1, in this example you should start the *Receiver* program before you start the *Sender* program (Why?).
- Upload the code on Canvas

Task 3
The same as in Task 2 (and also similar to 'Preparation for Lab1'), but instead extend the example to let communication pass to other computers.
Contact a co-student to negotiate over sending and receiving each other's information about the graphical objects. The only things that have to be taken care of here are to change the code according to the network IP addresses instead of running your programs locally. This you also should have done in 'Preparation for Lab1'.
To pass task 3:

- Explain the code, run the programs
- Upload the code on Canvas

After you finish Task 3 you are encouraged to experiment further with the code. Add more graphics description.

GOOD LUCK!!!