# CO3099-Foundations of Cyber Security-Lab 01

The purpose of this lab exercise is to refresh your knowledge on Java Programming. Please note, it is assumed that you have a basic understanding of Java programming. For understanding the principles of Java, please go through contents of the course: Java crash course

**Question 0:** Make sure you know how to compile and run Java programs, from whatever IDEs (IntelliJ is preferred for the set of labs in this module) you like or just from the command line using the javac and java commands.

You should already have a Java IDE installed in your PCs. Preferred IDE is IntelliJ for this lab. However, you can use other IDEs such as Eclipse which should also be installed in the Lab PCs.

Seek help from the Teaching Assistants in the lab if necessary.

Next, complete the following exercises:

**Question 1**. There are various streams you can use to read/write from different sources:

- FileInputStream/FileOutputStream - file
- DataInputStream/DataOutputStream - data types
- ObjectInputStream/ObjectOutputStream - any Java Object

You can also *wrap* streams around streams. Typically, we create data streams or object streams, around file streams or socket streams.

1. Open the DataStreamExample.java file and examine how it works. The program first opens a FileOutputStream to prepare for the writing of the "somefile" file. Then it wraps a DataOutputStream around it so that simple data types can be written to it. An integer and a string is written to the file.

2. Compile and run the program. Locate the file somefile in your current directory (where you run the java program). If you try to open it in a text editor, it will not result in completely readable text since it is encoded when Java writes the data types to it.

3. Now modify the program so that after writing, it immediately reads back the two data elements from the file, and display them on the screen. For this you will need methods from the classes FileInputStream and DataInputStream. Check the Java API to find out the

appropriate methods and how it works. Compile and run the program to make sure it produces the correct result.

**Question 2**. Client Server Socket Programming. A *socket* is a point of network connection between two computers over the network. It is specified by the hostname (or IP address) and a port number.

1. Inspect the SocketExampleClient.java and SocketExampleServer.java files and try to understand how they work. Basically, the server opens a "ServerSocket" that listens for incoming connections. The client connects to it. They create data streams around the socket (byte)streams so that subsequently they can use it to send/receive basic data types. The client sends two strings to the server using the writeUTF() method (UTF is a string encoding). The server receives and prints them.

2. Compile and run both programs (at different terminals, on the same machine) to see how they work. You need to start the server first, otherwise the client has nowhere to connect to. localhost stands for the local machine; since both programs are running on the same machine, this is fine.

3. Now modify the programs so that after receiving the strings, the server changes them to uppercase and send it back to the client. The client displays the received strings. You will need to check the String class in the Java API to find the method for uppercase conversion.

4. Modify the server program further so that it does not terminate after it serves one client; instead it waits for the next client to connect. (For simplicity you can ignore that a new client may want to connect while the current one is still being connected; i.e. you do not need to consider the use of threads.)

Solutions:

Question 1: DataStreamExample.java

Question 2: SocketExampleClient.java, SocketExampleServer.java