RIT | Golisano College of Computing and Information Sciences
School of Information

ISTE-121
Lab07 / HW06 - Object IO over Sockets

NOTE:

This is a combined Lab and HW. This means you will receive both Lab and HW credit for this assignment. When done, submit your work to **both** the Lab07 and HW06 Assignment folders. The lab portion will be assessed as usual (10 points) and the homework portion will be assessed against the gradesheet at the end of this document.

Objectives:

Write a Multi Threaded Client / Server GUI that sends information using Object IO. The scenario we will be using is one of several people entering order information information on the client computers and this information being sent to the server for collection and storage.

You can write this lab on your own, or it can be written as a team of two people (**recommended**). One person writes the client, the other writes the server. With two people, you must coordinate communication protocols to ensure reads and writes are happening correctly.

Details

Consider a scenario where in-store associates are placing orders for customers. Each order is for some number of a single item. To place the order we need:

- Customer’s name
- Customer’s address
- Customer’s email address
- Item number
- Quantity

Create a class (called Order) with attributes for each of the above pieces of information and appropriate constructors, accessors, and mutators. When an associate places an order, the client will create an object of class Order and send it, as an object, to the server. The server will then record this in a Vector or ArrayList of Orders.

Serializability

For an object to be sent over a socket, using ObjectOutputStream / ObjectInputStream, it must be serializable. This means, that the Order class (above) must implement the Serializable interface. This, in turn, means that the class must have a declaration such as:

```
   private static final long serialVersionUid = 01L;
```

in the attributes. This is a version number. As you change your class, you can update this number. This allows the receiver to check to see if it is using the same version of the Order class.

You do **not** have to keep this updated. Set it to 1 and leave it alone. However, you **must** have it in your class for object IO over sockets to work correctly!

The Server

The server will react to the **CloseRequest** event by terminating all connections from associates, and writing the Orders to a file (named **orders.obj**). When the server starts, it will check for this file. If it does not exist, it will start with an empty list. Otherwise, it will read this list in as a starting point to which to add new orders.

The Client

The sales associate, via the client, once connected to the server, can send one or many orders during their session. When done (the associate clicks a button or menu item) the client will send a message to the server saying that it is done. The server will terminate the client's thread in response to this.

What to send

The client will send only objects to the server (remember **ObjectOutputStream** and **ObjectInputStream**?). But objects of different classes may be sent. An object of class Order seems likely, but you can also send String objects, Integer objects, etc. As long as the receiver knows what type of object will be sent, or can determine it via **instanceof**, all will be well.

You need to design a protocol that will do all you need to do, sending only objects. The protocol must specify what types of objects to send when.

Client Requirements

As either a button or menu item, the associate using your client program must be able to send to the server a command to:

- Record a new order. The order must be sent to the server and the server must reply with an acknowledgement that the order was received. Some visual representation of this acknowledgement must be presented to the user.
- Request the number of orders currently in the list. The server will respond with a message that includes that number. It will be displayed to the user in a pop-up window (Alert).
- Terminate the connection to the server. This will send some message to the server telling it to terminate the client's thread. There is no reply. THIS SHOULD OCCUR WHEN THE ASSOCIATE CLICKS A BUTTON OR CLOSES THE WINDOW.

Server Requirements

The server must be able to be started or stopped. When stopped, no new connections are allowed. When started, new connections are permitted. Existing connections continue to operate, uninterrupted, even when the server is stopped.

The server must be able to be closed (e.g., the **CloseRequest** event again) by closing its window. When it is closed, it must terminate all client connections. Clients must handle this (perhaps by detecting an exception the next time they try to read or write the socket or its streams). Also, before exiting, the server must write the list of Orders to the file **orders.obj**.

When the server starts, it must look for **orders.obj** and, if found, read the Orders into its list of Orders. If it is not found, start with an empty list.

The server's GUI must display the IP address of the machine it is running on.

The server's GUI must display the current number of orders in the list.

The server must have a button or menu item to **convert the orders**. This means writing all orders in the list to a file named **orders.csv** in comma separated format. Each Order will occupy one line in this file, and must be formatted as:

```
   "Customer's name","Customer's address","Customer's email address",Item_number,Quantity
```

NOTE: the first 3 fields are quoted.
ALSO: once the csv file is written and closed, the list is to be cleared (emptied).

BEWARE WEIRD JAVA Object IO ISSUE
It turns out, when you open the Socket's input stream as an ObjectInputStream, Java reads ahead right away, without waiting, to see if the stream is at End Of File. If there is no input on that stream, your program will block.

If the process at the other end of the Socket has already opened its ObjectOutputStream, this blocking does not occur. There is a short 'hand shake' between the two processes to avoid this.

What does this mean for you? When using Object IO across Sockets, **always open the ObjectOutputStream first**, then the ObjectInputStream. If both parties do this, it guarantees that nobody gets stuck waiting for the other guy.

BEWARE SYNCHRONIZATION ISSUES
- When client threads add Orders to the list.
- When the user chooses to convert orders to csv format (above), perhaps while client threads are working.
- When clients update the number of orders displayed on the server.
- Etc.

Test, Test, Test
We want this code to be as bullet-proof as possible. You should trap every possible exception and tell the user it happened and roughly where in the code (e.g., the method name). You should test this out with one and also many simultaneous clients.

HINT: You may want to use the log() and alert() methods from Day 20's lecture notes in your server where threading is present.

NOTE: My solution to this is available as two jarfiles in today's downloads. You can run these to see how things are supposed to look and work.

Submit you *.java file(s) to both the Lab07 and HW06 Assignment folders when your code is working properly.

ISTE-121 HW06 Gradesheet
Object IO Over Sockets

Criteria	**Possible points**	**Earned points**
Order Class		
Separate order class exists	5	
Order is serializable	2	
Contains attributes for name, address, email, item number, quantity	3	
Subtotal (Order class)	**10**	
Client		
When terminated by server, clients disconnect gracefully	10	
When client records a new order, it works, and a acknowledgement is received	5	
When client requests the # of orders, the server supplies it and it is displayed	5	
When client window is closed, or a terminate button/menu item is chosen, the connection to the server is closed	5	
Client meets all program requirements	15	
Subtotal (client)	**40**	
Server		
Can be started and stopped	5	
When stopped existing clients continue to operate	5	
When window is closed, all clients are terminated	5	
When window is closed, all orders are written to orders.obj	5	
When started, server reads orders (if any) from orders.obj	4	
Server GUI displays server's IP	2	
Server GUI displays current # of orders	2	
When the list is converted, it is written to orders.csv	5	
When the list is converted, the server's order list is cleared	2	
Server meets all program requirements	15	
Subtotal (server)	**50**	
Total points earned:	**100**	

ISTE-121 HW06 Gradesheet (cont.)
Object IO Over Sockets

Deduction violations after above grading		
-Xlint messages. Need a clean compile -2		
Deduction for program not following naming conventions		
Deduction for proper coding style not being used: indentation, use of white space for readability		
Deduction for missing JavaDoc documentation, File & methods -5		
Server contains inadequate in-code documentation -3		
Total Grade:	**100**	

Additional Comments: