ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Προχωρημένα Θέματα Βάσεων Δεδομένων

Auction Project Documentation

Λώλος Κωνσταντίνος - 03110004

Μπουρίκας Φοίβος -

Ποδηματά Χαρίκλεια -

9ο Εξάμηνο ΣΗΜΜΥ

22 Μαΐου 2015

Περιεχόμενα

1 Εισαγωγή

Σε αυτό το project κληθήκαμε να υλοποιήσουμε την επικοινωνία μεταξύ δύο servers και πολλών clients. Στόχος μας ήταν να δημιουργήσουμε μία εφαρμογή για δημοπρασίες. Για τους σκοπούς του project χρησιμοποιήσαμε threads και channel, για την επικοινωνία. Η γλώσσα που χρησιμοποιήσαμε για την υλοποίηση του project ήταν η `Java`.

Για τους σκοπούς της άσκησης, τρέχαμε δύο ξεχωριστές `Databases` (μία σε κάθε έναν auctioneer) και οι μόνες πληροφορίες που αντάλλασσαν μεταξύ τους οι servers ήταν οι πληροφορίες του τρέχοντος αντικειμένου, και του αντίστοιχου bidder. Οι auctioneers τρέχουν ως δύο διαφορετικά threads στην ίδια διεργασία.

Οι δημοπρασίες είναι πωλήσεις στις οποίες δεν υπάρχει κάποια καθορισμένη τιμή για κάποιο αντικείμενο. Αντιθέτως, ο auctioneer λαμβάνει bids για κάθε αντικείμενο. Τα bids αυτά πρέπει να είναι αυστηρά αυξανόμενα, δηλαδή, αν το τελευταίο bid ήταν 25 δολάρια, το αμέσως επόμενο bid θα πρέπει να είναι μεγαλύτερο από 25 δολάρια. Η μόνη εξαίρεση σε αυτό, είναι η αρχική τιμή. Τα bids επιπλέον, είναι ακέραιες μεταβλητές. Ο auctioneer συνεχίζει να δέχεται bids από τους ενδιαφερόμενους bidders έως ότου περάσει κάποιος συγκεκριμένος χρόνος. Όταν περάσει αυτό το χρονικό διάστημα, ο bidder με το τελευταίο bid κερδίζει το αντικείμενο.

Στη συνέχεια αυτού του εγγράφου περιγράφεται εκτενώς τόσο ο τρόπος που ένας χρήστης μπορεί να συνδεθεί και να χρησιμοποιήσει την εφαρμογή μας, καθώς και όλη η αρχιτεκτονική του συστήματός μας.

2 User Documentation Manual

Σε αυτή την ενότητα θα περιγράψουμε πώς μπορεί ένας χρήστης να χρησιμοποιήσει την εφαρμογή μας, είτε αυτός ο χρήστης θέλει να τρέξει τον server, ή τους bidders.

2.1 Server

Για να τρέξουμε τον server, αρκεί να γράψουμε σε ένα αρχείο `.txt` τις πληροφορίες για τα αντικείμενα που θέλουμε να δώσουμε στο auction. Η μορφή αυτού του αρχείου θα είναι:

```
<value of L, in milliseconds>
<number of items (N)>
<initial price of item 1> <description of item 1>
<initial price of item 2> <description of item 2>
.
.
.
<initial price of item N> <description of item N>
```

Το αρχείο αυτό θα δοθεί ως είσοδος στον serverLauncher. Έτσι, για να εκκινήσουμε και τους δύο Servers σε δύο διαφορετικές ports, αρκεί να πληκτρολογήσουμε:

```
java ServerLauncher <auct_conf.txt>
```

Μόλις πατηθεί αυτή η εντολή, έχει ξεκινήσει η λειτουργία των δύο Servers και ο χειριστής δεν χρειάζεται να κάνει κάτι άλλο. Οι Servers θα εγκαταστήσουν την επικοινωνία τους και θα αναλάβουν να στείλουν τα κατάλληλα control messages στους bidders που θα συνδεθούν. Περισσότερες πληροφορίες για τις λειτουργίες του κώδικα, θα βρείτε παρακάτω.

2.2 Client

Προκειμένου να τρέξουμε έναν bidder, το μόνο που χρειάζεται να κάνουμε είναι να γράψουμε την εντολή:

```
java ClientLauncher <host> <port> <bidderName>
```

Όπου `<host>` είναι ένα string, για παράδειγμα `localhost`, port είναι μία από τις ports που είναι διαθέσιμες για τους δύο auctioneers (δηλαδή, μία εκ των 4444, 5556) και `bidderName`, το όνομα με το οποίο θέλει να συνδεθεί ο bidder.

Αν η σύνδεσή του είναι επιτυχής, θα του εμφανιστούν κατάλληλα μηνύματα στο terminal. Παραθέτουμε παρακάτω τις εντολές που μπορεί να πληκτρολογήσει ο χρήστης:

- `i_am_interested`: Όταν ο χρήστης δει ένα νέο αντικείμενο, τότε έχει τη δυνατότητα πληκτρολογώντας αυτή την εντολή να δηλώσει το ενδιαφέρον του και να ενημερώνεται στη συνέχεια για αυτό το προϊόν και να δώσει και bid. Αν το πληκτρολογήσει σε κάποια στιγμή που δεν έχει ζητηθεί από τους Servers να δηλωθεί το ενδιαφέρον για κάποιο αντικείμενο, τότε ο χρήστης ενημερώνεται με κατάλληλο μήνυμα.

- `quit`: Ο χρήστης μπορεί κάθε στιγμή να πληκτρολογήσει `quit` προκειμένου να βγει από την εφαρμογή και να κλείσει τη σύνδεσή του με τους Servers. Σε αυτή την περίπτωση, προτού κλείσει το κανάλι του, ο χρήστης ενημερώνεται σχετικά με τα προϊόντα που προμηθεύτηκε από τις δημοπρασίες.

- `list_high_bid`: Ο χρήστης μπορεί κάθε στιγμή να πληκτρολογήσει `list_high_bid` προκειμένου να του εμφανιστεί στην οθόνη του το μεγαλύτερο bid για το τρέχον αντικείμενο. Να διευκρινιστεί εδώ όταν ο χρήστης δεν είναι στις λίστες των ενδιαφερόμενων, δεν θα του εμφανιστεί αυτή η πληροφορία και θα ενημερωθεί με κατάλληλο μήνυμα.

- `list_description`: Ο χρήστης μπορεί κάθε στιγμή να πληκτρολογήσει

  `list_description` προκειμένου να του εμφανιστεί στην οθόνη του η περιγραφή για το τρέχον αντικείμενο. Να διευκρινιστεί εδώ όταν ο χρήστης δεν είναι στις λίστες των ενδιαφερόμενων, δεν θα του εμφανιστεί αυτή η πληροφορία και θα ενημερωθεί με κατάλληλο μήνυμα.

- `bid <amount>` : Ο χρήστης μπορεί να ποντάρει για να κερδίσει το αντικείμενο. Έτσι, πληκτρολογώντας `bid <amount>` δίνει μία νέα τιμή για το τρέχον αντικείμενο. Αν η τιμή που δώσει είναι μικρότερη της τρέχουσας, τότε το bid του δεν γίνεται αποδεκτό και ενημερώνεται με κατάλληλο μήνυμα. Αν η τιμή που δώσει είναι μεγαλύτερη και γίνει τελικώς αποδεκτή και πάλι ενημερώνεται με κατάλληλο μήνυμα. Να διευκρινιστεί εδώ όταν ο χρήστης δεν είναι στις λίστες των ενδιαφερόμενων, δεν θα του εμφανιστεί αυτή η πληροφορία και θα ενημερωθεί με κατάλληλο μήνυμα.

Περισσότερες πληροφορίες σχετικά με τον κώδικα της εφαρμογής μας θα παραθέσουμε στην επόμενη ενότητα.

3 Κύριες σχεδιαστικές αποφάσεις

3.1 Περιγραφή του κώδικα

3.1.1 Κώδικας των Servers

ServerLauncher.

Το ServerLauncher είναι το πρόγραμμα το οποίο τρέχουμε για να αρχικοποιήσουμε τους Servers. Παίρνει ως είσοδο ένα αρχείο, το οποίο έχει καταχωρημένα τα προϊόντα για τα auctions που θα γίνουν. Η κλάση αυτή, ανοίγει 2 ports για τους 2 auctioneers που θα τρέξουν. Οι 2 auctioneers τρέχουν ως threads της ίδιας διεργασίας (της ServerLauncher), η οποία αναλαμβάνει να τους κάνει και start.

Οι δύο auctioneers έχουν μεγάλο βαθμό ανεξαρτησίας μεταξύ τους, ώστε με ελάχιστες αλλαγές στις ρυθμίσεις τους, να μπορούν να λειτουργήσουν σε ξεχωριστά μηχανήματα. Επικοινωνούν μεταξύ τους με μηνύματα που ανταλλάσσουν μέσω sockets.

Παρακάτω παρατίθεται ο κώδικας της κλάσης.

```
import java.io.IOException;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;



public class ServerLauncher
{
        static final int callerPort = 4444;
        static final int calleePort = 5556;

        public static void main(String[] args) throws IOException
        {
                String confFile = getConfFile(args);
                if (confFile == null)
                        return;

                Lock debugLock    = new ReentrantLock();
                Auctioneer caller = new Auctioneer(callerPort, debugLock);
                Auctioneer callee = new Auctioneer(calleePort, debugLock);
                caller.makeCaller(calleePort);

                caller.configure(confFile);
                callee.configure(confFile);
                caller.start();
                callee.start();
```

```
                try {
                        caller.join();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                try {
                        callee.join();
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                System.exit(0);
        }

        private static String getConfFile(String[] args)
        {
                if (args.length != 1) {
                        System.out.println("Usage: java ServerLauncher <configuration file>");
                        return null;
                } else {
                        return args[0];
                }
        }
}
```

Auctioneer.

Η κλάση Auctioneer είναι ο κώδικας που τρέχουν οι 2 servers. Οι δύο auctioneers ενημερώνονται για την κατάσταση στην οποία βρίσκεται ο άλλος auctioneer (λ.χ αν έχει δεχτεί νέο bid ή αν είναι έτοιμος να δεχτεί bids κλπ) μέσω των μηνυμάτων που παίρνουν. Οι auctioneers προκειμένου να συντονιστούν, χρησιμοποιούν αυτό το state τους για να ελέγξουν ότι και οι δύο είναι στη σωστή κατάσταση για την κάθε ενέργεια. Ο κάθε auctioneer είναι ένα thread που έχει κοινό lock για το loop του proxy και το loop του timer. Έτσι, οι μέθοδοι των auctioneers δεν γίνεται να εκτελεστούν παράλληλα, αλλά περιμένουν να πάρουν το lock. Ο κάθε auctioneer προκειμένου να μιλήσει με τον άλλον auctioneer, αλλά και προκειμένου να μιλήσει με τους bidders χρησιμοποιεί έναν proxy. Για την κλάση του proxy και τις λειτουργίες που επιτελεί θα αναφερθούμε παρακάτω.

Αξίζει επίσης, να σημειωθεί η διαδικασία με την οποία χειριζόμαστε το καινούριο bid που μου έρχεται από τον client. Αυτό που κάνουμε είναι το εξής: έχουμε currentBid το οποίο συμβολίζει το bid που έχει συμφωνηθεί από τους δύο auctioneers και tempBid το οποίο είναι το bid που έχει δει κάθε ένας auctioneer, αλλά δεν το έχει «συμφωνήσει» ακόμη με τον άλλον auctioneer. Επιπλέον, αν τα tempBids και των δύο auctioneers είναι ίδια, εμείς θα καταχωρήσουμε ως holder τον bidder που προέρχεται από τον caller.

Θα αναλύσουμε τώρα τις μεθόδους που χρησιμοποιούνται από τους auctioneers.

- **makeCaller**: Μέθοδος που χρησιμοποιείται για να αλλάξουμε τον τύπο ενός auctioneer και να τον κάνουμε από callee, caller. Αυτοί οι δύο τύποι, είναι μία σύμβαση που χρησιμοποιήσαμε προκειμένου να ονοματίσουμε τους δύο auctioneers και να δώσουμε μία μικρή ιεραρχία για τις περιπτώσεις των conflicts. Στην ιεραρχία αυτή, ο caller auctioneer έχει πάντα προτεραιότητα.

- **configure**: Μέθοδος που χρησιμοποιείται για την αρχικοποίηση του DBServer (περισσότερα για αυτή την κλάση θα αναφέρουμε αργότερα) και για το parsing της εισόδου.

- **runCountDown**: Μέθοδος που χρησιμοποιείται για να χρονομετρήσουμε μία αντίστροφη μέτρηση έως ότου αρχίσουν οι auctions. Παράλληλα, στέλνουμε μέσω του proxy σε όλους τους bidders που έχουν γίνει connected το μήνυμα σχετικά με το πόσος χρόνος μένει μέχρι την έναρξη.

- **beginAuctions**: Μέθοδος για να αρχίσουμε τη διαδικασία των auctions. Στέλνουμε αρχικά μήνυμα μέσω του proxy σε όλους τους συνδεδεμένους χρήστες και στη συνέχεια ελέγχει ο κάθε auctioneer την κατάσταση του άλλου, για να δει αν και οι δύο βρίσκονται στο state READY_TO_BEGIN τότε προχωράμε στο επόμενο auction.

- **beginNextAuction**: Μέθοδος για να ξεκινήσουμε την επόμενη auction. Αν δεν υπάρχουν πλέον items, στέλνουμε το μήνυμα Auctions have ended σε όλους τους συνδεδεμένους χρήστες. Αλλιώς, το επόμενο αντικείμενο γίνεται το τρέχον, αλλάζουμε τα states των δύο auctioneers και περιμένουμε τους clients για κάποιο χρονικό διάστημα να δηλώσουν ότι ενδιαφέρονται για να τους εγγράψουμε στη λίστα των ενδιαφερόμενων χρηστών για το αντικείμενο.

- **interestTimerEnded.** Μέθοδος για να χειριστούμε το ότι τελείωσε ο χρόνος που είχαν οι χρήστες για να δηλώσουν ενδιαφέρον για κάποιο αντικείμενο. Αν κανένας χρήστης δεν δήλωσε ενδιαφέρον, τότε το αντικείμενο γίνεται discarded επειδή δεν υπάρχει ενδιαφέρον. Αν υπάρχουν ενδιαφερόμενοι χρήστες, τότε στέλνουμε το μήνυμα ότι μπορούν να αρχίσουν να δίνουν bids για το εν λόγω αντικείμενο, αλλά *μόνο σε όσους χρήστες έχουν δηλώσει ότι ενδιαφέρονται.*

- **setBidTimer**: Μέθοδος για να αρχικοποιήσουμε ένα χρονόμετρο ώστε να περιμένουμε τους χρήστες να δώσουν τα bids τους. Παράλληλα, αλλάζουμε και το state των δύο auctioneers.

- **bidTimerEnded**: Μέθοδος για να εκτελέσουμε τις ενέργειες που απαιτούνται αφότου τελειώσει ο διαθέσιμος χρόνος για να κάνουν bid οι ενδιαφερόμενοι χρήστες. Αρχικά, ελέγχουμε αν και ο άλλος auctioneer είναι στην κατάσταση READY_TO_END, τότε κάνουμε anounceWinner σε όλους τους ενδιαφερόμενους χρήστες και προχωράμε στο επόμενο αντικείμενο. Στην περίπτωση που κανένας εκ των ενδιαφερόμενων χρηστών δεν έχει κάνει bid για το αντικείμενο, κάνουμε discount 10% και ξανατρέχουμε τον timer. Επαναλαμβάνουμε αυτή τη διαδικασία με το discount για 5 φορές και αν κανείς δεν έχει ενδιαφερθεί, κάνουμε discard το αντικείμενο. Επιπλέον, χειριζόμαστε το εξής σενάριο: Έστω ότι για τον auctioneer 1 έχει τελειώσει ο χρόνος, αλλά για τον auctioneer 2 όχι και αυτός λαμβάνει ένα νέο bid. Τότε, επαναρχικοποιούμε το χρόνο και αν ο auctioneer 1 είχε λάβει και αυτός εκπρόθεσμο bid το κρατάμε προσωρινά αποθηκευμένο. Όταν λοιπόν, μας στείλει το bid ο auctioneer 2, τότε κάνω handle και τα προσωρινώς αποθηκευμένα, αλλιώς τα κάνω discard.

- **anounceWinner**: Μέθοδος με την οποία παίρνουμε τον νικητή από το τρέχον item και στη συνέχεια στέλνουμε στους ενδιαφερόμενους χρήστες το μήνυμα stop_bidding, το οποίο περιέχει και το όνομα του νικητή. Τέλος, ενημερώνουμε την βάση με το νικητήριο bid.

- **broadcastInfo**: Μέθοδος με την οποία στέλνουμε ένα info message σε όλους τους χρήστες μέσω του proxy.

- **processMessage**: Μέθοδος για την επεξεργασία των μηνυμάτων που δεχόμαστε. Διαβάζουμε από το channel και φροντίζουμε να αποθηκεύσουμε και τα υπόλοιπα μηνύματα σε περίπτωση που μας έχουν έρθει δύο μηνύματα ταυτόχρονα.

- **handleMessage**: Μέθοδος η οποία καλείται αμέσως μόλις αναγνωριστεί κάποιο μήνυμα. Παίρνουμε την εντολή από το μήνυμα και στη συνέχεια την κάνουμε handle με την κατάλληλη μέθοδο.

- **handleQuit**: Μέθοδος η οποία όταν κάποιος χρήστης πατήσει «quit», του κλείνει τη σύνδεση στο channel και επιπλέον, τον σβήνει από τη λίστα με τους ενδιαφερόμενους

χρήστες (αν είναι σε αυτή τη λίστα).

- **handleGotBid**: Μέθοδος για το χειρισμό ενός νέου bid από τον bidder. Μόλις λάβουμε το bid, στέλνουμε got bid στον άλλον auctioneer. Αν αυτός μας στείλει πίσω bidOk, τότε κάνουμε update το highest bid. Αν όμως δεν μας στείλει πίσω bidOk, τότε αυτό σημαίνει ότι μου έχει στείλει ο ίδιος κάποιο μεγαλύτερο bid, το οποίο πρέπει να καταχωρηθεί ως το highest.

- **updateHighestBid**: Μέθοδος για το χειρισμό καινούριου highest bid. Μόλις λάβουμε τέτοιο μήνυμα, ενημερώνουμε μέσω του proxy όλους τους ενδιαφερόμενους χρήστες ότι έχουμε καινούριο highest bid και ξαναθέτουμε τον timer για το bidding.

- **handleBid**: Μέθοδος για τον χειρισμό ενός νέου bid από τον χρήστη. Αν ο auctioneer είναι στην κατάσταση ACCEPTING_BIDS τότε, με το που το λάβουμε, στέλνουμε το μήνυμα και στον άλλον auctioneer για να ακολουθηθεί η διαδικασία που περιγράψαμε παραπάνω στην handleGotBid μέθοδο.

- **handleReadyToEnd**: Μέθοδος για τον χειρισμό της κατάστασης READY_TO_END. Αν δεν έχει δώσει κανείς bid, τότε, κάνουμε discount όπως έχει περιγραφεί παραπάνω, αλλιώς ανακοινώνουμε τον νικητή, αλλάζουμε το state του auctioneer και προχωράμε στην επόμενη auction.

- **handleReadyToRun**: Αν και οι δύο auctioneers είναι READY_TO_RUN, τότε ξεκινάμε την επόμενη auction.

- **handleInterest**: Αν κάποιος χρήστης δηλώσει ενδιαφέρον για το αντικείμενο, τότε τον προσθέτουμε στη λίστα με τους interested users για το συγκεκριμένο αντικείμενο.

- **handleConnect**: Αν κάποιος χρήστης συνδεθεί στο κανάλι μας, τότε τον προσθέτουμε στη λίστα με τους ενεργούς χρήστες.

Παρακάτω παρατίθεται και ο κώδικάς μας για τους Auctioneers:

```
import java.io.IOException;
import java.nio.channels.SelectionKey;
import java.nio.channels.SocketChannel;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.Map;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;


public class Auctioneer extends Thread
{
    // Constants
    public static final String clientChannel    = "clientChannel";
    public static final String serverChannel    = "serverChannel";
    public static final String channelType      = "channelType";
    public static final String bytesReadString  = "bytesRead";
    public static final String clientName       = "clientName";
    public static final String peerName          = "__peer";

    private enum State { OFFLINE, READY_TO_BEGIN, ACCEPTING_INTERESTS,
            WAITING_INTERESTED_COUNT, ACCEPTING_BIDS, READY_TO_END }

    // private fields
    private final int countdownTime = 5;
    private int countDownInterval = 5;
    private int timeLapse;
    private ArrayList<Item> items;
    private String type = "callee";
    private Proxy proxy;
    private Lock lock;
    private Lock debugLock;
    private MessageFactory messageFactory;
    private DBServer dbServer;
    private Item currentItem;
    private State state;
    private State peerState;
    private Timer timer;
    private boolean auctionsEnded;


    public Auctioneer(int port, Lock debugLock)
    {
        lock           = new ReentrantLock();
```

```
        this.debugLock = debugLock;
        proxy          = new Proxy(port, this);
        messageFactory = new MessageFactory();
        auctionsEnded  = false;

        peerState = State.OFFLINE;
        state     = State.OFFLINE;
    }

    /**
     * Set the port of the callee auctioneer and convert this one into a caller
     * @param peerPort
     */
    public void makeCaller(int peerPort)
    {
        this.type = "caller";
        proxy.setPeerPort(peerPort);
    }

    /**
     * Instantiates a DBServer and parses the configuration file
     * @throws IOException
     */
    public void configure(String confFile) throws IOException
    {
        if (type.equals("caller"))
            dbServer = DBServer.getCallerInstance();
        else
            dbServer = DBServer.getCalleeInstance();
        dbServer.setAuctioneer(this);

        ConfParser confParser = new ConfParser(confFile);
        confParser.parse();
        timeLapse = confParser.getTimeLapse();
        items = confParser.getItems();
        dbServer.initAuctions(items);
    }

    /**
     * @throws IOException
     */
    public void run()
    {
        runCountDown(countdownTime);


        try {
            proxy.run();
        } catch (IOException e) {
            e.printStackTrace();
```

```
        }

        debug("Shuting Down ...");
    }

    /**
     * Runs a countdown till the auctions start
     * @param seconds
     */
    private void runCountDown(int seconds)
    {
        final int remaining = seconds - countDownInterval;
        Timer t = new Timer();
        t.schedule(new TimerTask() {
            @Override
            public void run() {
                lock.lock();
                if (remaining <= 0) {
                    beginAuctions();
                } else {
                    String countdownMessage = "Auctions start in "
                                            + remaining + " seconds!";
                    broadcastInfo(countdownMessage);
                    if (type.equals("caller"))
                        debug(countdownMessage);
                    runCountDown(remaining);
                }
                lock.unlock();
            }
        }, countDownInterval * 1000);
    }

    /**
     * Begins the auction process
     * This method is to be called holding the lock
     */
    private void beginAuctions()
    {
        debug("Auctions are beggining!");
        broadcastInfo("Welcome to the auction house!");

        setState(State.READY_TO_BEGIN);
        Message m = messageFactory.createMessage(Constants.ready_to_run);
        proxy.send(Auctioneer.peerName, m);

        if (peerState == State.READY_TO_BEGIN) {
            beginNextAuction();
        }
    }
```

```
/**
 * Begins the next auction
 * This method is to be called holding the lock
 */
private void beginNextAuction()
{
    if (timer != null)
        timer.cancel();

    if (items.isEmpty()) { // there are no more items to auction
        Message m = messageFactory.createMessage(Constants.auction_complete);
        proxy.broadcast(m);
        debug("Auctions have ended");
        auctionsEnded = true;
        return;
    }

    debug("Starting Auction for next item");
    currentItem = items.remove(0); // pop the first item
    int itemId = currentItem.getId();
    String description = currentItem.getDescription();
    int startingPrice = currentItem.getStartingPrice();
    Message m = messageFactory.createBidItemMessage(itemId, description, startingPrice);
    proxy.broadcast(m);
    setState(State.ACCEPTING_INTERESTS);
    peerState = State.ACCEPTING_INTERESTS;

    timer = new Timer();
    timer.schedule(new TimerTask() {
        @Override
        public void run() {
            lock.lock();
            interestTimerEnded();
            lock.unlock();
        }
    }, timeLapse * 1000);
}

protected void interestTimerEnded()
{
    setState(State.WAITING_INTERESTED_COUNT);
    int count = currentItem.getInterestedUsers().size();
    Message toPeer = messageFactory.createInterestedCountMessage(count);
    proxy.send(Auctioneer.peerName, toPeer);
    debug("interest time ended!");
    if (State.WAITING_INTERESTED_COUNT == peerState) {
        if ((count == 0) && (currentItem.getPeerInterestedCount() == 0)) {
            debug("Item " + currentItem.getId() +
                                " is discarded due to lack of interest");
            beginNextAuction();
```

```
            } else {
                peerState = State.ACCEPTING_BIDS;
                Message m =
                    messageFactory.createStartBiddingMessage(currentItem.getStartingPrice(),
                                                             currentItem.getId());
                proxy.broadcast(currentItem.getInterestedUsers(), m);
                setBidTimer();
            }
        }
    }

    private void setBidTimer()
    {
        setState(State.ACCEPTING_BIDS);

        if (timer != null)
            timer.cancel();

        timer = new Timer();
        timer.schedule(new TimerTask() {
            @Override
            public void run() {
                lock.lock();
                bidTimerEnded();
                lock.unlock();
            }
        }, timeLapse * 1000);
    }

    protected void bidTimerEnded()
    {
        debug("bid timer ended");
        setState(State.READY_TO_END);

        Message m = messageFactory.createMessage(Constants.ready_to_end);
        proxy.send(Auctioneer.peerName, m);

        // if peer's bidTimer has already ended and there is no tempBid
             // for which we are expecting confirmation
        if ((peerState == State.READY_TO_END)
                && (currentItem.getTempBid() == currentItem.getCurrentBid())){
            currentItem.incrDiscountRound();
            int discountRound = currentItem.getDiscountRound();
            if (currentItem.getCurrentBidder().equals(Constants.no_holder)
                                                        && (discountRound < 5)) {
                int newPrice = (currentItem.getStartingPrice()
                                                        * (10 - discountRound)) / 10;
                currentItem.setCurrentBid(newPrice);
                currentItem.setTempBid(newPrice);
                peerState = State.ACCEPTING_BIDS;
```

```
                Message bidAgain = messageFactory.createNewHighBidMessage(
                                                    Constants.no_holder, newPrice);
                proxy.broadcast(currentItem.getInterestedUsers(), bidAgain);
                debug("sent bid again message");
                setBidTimer();
            } else {
                anounceWinner();
                peerState = State.ACCEPTING_INTERESTS;
                beginNextAuction();
            }
        }
    }

    private void anounceWinner()
    {
        int itemId = currentItem.getId();
        String winner = currentItem.getCurrentBidder();
        int winningBid = currentItem.getCurrentBid();
        Message m = messageFactory.createStopBiddingMessage(itemId, winner, winningBid);
        proxy.broadcast(currentItem.getInterestedUsers(), m);
        dbServer.updateBid(currentItem);
    }

    private void broadcastInfo(String message)
    {
        Message m = messageFactory.createInfoMessage(message);
        proxy.broadcast(m);
    }

    public void processMessage(String read, SelectionKey key)
    {
        @SuppressWarnings("unchecked")
        Map<String, String> clientProps = (Map<String, String>) key.attachment();
        String total = clientProps.get(Auctioneer.bytesReadString) + read;

        ArrayList<Message> messages = messageFactory.extractMessages(total);
        String remaining = messageFactory.getRemaining();
        clientProps.put(Auctioneer.bytesReadString, remaining);

        for (Message m : messages)
            handleMessage(m, key);
    }

    /**
     * Takes the actions necessary in response to the given message
     * @param message
     * @param name
     * @param channel
     */
    private void handleMessage(Message message, SelectionKey key)
```

```
    {
        String command = message.getCommand();
        debug("Received: " + message);

        @SuppressWarnings("unchecked")
        Map<String, String> clientProps = (Map<String, String>) key.attachment();
        String name  = clientProps.get(Auctioneer.clientName);
        SocketChannel channel = (SocketChannel) key.channel();

        switch (command) {

        // message from bidder
        case Constants.connect:
            handleConnect(name, message, channel, clientProps);
            break;
        case Constants.i_am_interested:
            handleInterest(name, message);
            break;
        case Constants.my_bid:
            handleBid(name, message);
            break;
        case Constants.quit:
            handleQuit(name);
            break;

        // messages from peer
        case Constants.ready_to_run:
            handleReadyToRun();
            break;
        case Constants.ready_to_end:
            handleReadyToEnd();
            break;
        case Constants.got_bid:
            handleGotBid(message);
            break;
        case Constants.bid_ok:
            handleBidOk(message);
            break;
        case Constants.interested_count:
            handleInterestedCount(message);
            break;
        }
    }

    private void handleInterestedCount(Message message)
    {
        int peerCount = Integer.parseInt(message.getProperty(Constants.amount));
        currentItem.setPeerInterestedCount(peerCount);
        peerState = State.WAITING_INTERESTED_COUNT;
        if (State.WAITING_INTERESTED_COUNT == state) {
```

```
            if ((peerCount == 0) && (currentItem.getInterestedUsers().size() == 0)) {
                debug("Item " + currentItem.getId() +
                            " is discarded due to lack of interest");
                beginNextAuction();
            } else {
                peerState = State.ACCEPTING_BIDS;
                Message m = messageFactory.createStartBiddingMessage(
                        currentItem.getStartingPrice(), currentItem.getId());
                proxy.broadcast(currentItem.getInterestedUsers(), m);
                setBidTimer();
            }
        }
    }

    private void handleQuit(String userName)
    {
        proxy.closeConnection(userName);
        removeUserFromInterested(userName);
    }

    private void handleGotBid(Message message)
    {
        int amount = Integer.parseInt(message.getProperty(Constants.amount));
        String userName = message.getProperty(Constants.username);
            // give priority to the caller bids if amounts are equal
        if ((type.equals("caller") && amount > currentItem.getTempBid()) ||
            (type.equals("callee") && amount >= currentItem.getTempBid())) {
                updateHighestBid(userName, amount);
                Message m = messageFactory.createBidOkMessage(userName, amount);
                proxy.send(Auctioneer.peerName, m);
                for (String name: currentItem.getPendingBids().keySet())
                    handleBid(name, currentItem.getPendingBids().get(name));
        }
    }

    private void handleBidOk(Message message)
    {
        int amount = Integer.parseInt(message.getProperty(Constants.amount));
        String userName = message.getProperty(Constants.username);
        updateHighestBid(userName, amount);
    }

    private void updateHighestBid(String userName, int amount)
    {
        // temp bid needs to always be >= current bid
        if (amount > currentItem.getTempBid())
            currentItem.setTempBid(amount);
        currentItem.setCurrentBid(amount);
        currentItem.setCurrentBidder(userName);
```

```
        Message m = messageFactory.createNewHighBidMessage(userName, amount);
        proxy.broadcast(currentItem.getInterestedUsers(), m);
        debug("New highest bid from user " + userName + ", amount = " + amount);
        setBidTimer();
    }


    private void handleBid(String userName, Message message)
    {
        int amount = Integer.parseInt(message.getProperty(Constants.amount));
        int id = Integer.parseInt(message.getProperty(Constants.item_id));

        if (currentItem == null) {  // no item being auctioned at the moment
            Message m = messageFactory.createInfoMessage(
                        "No item being auctioned at the moment.");
            proxy.send(userName, m);
            return;
        }
        if (currentItem.getId() != id) {  // wrong item id - should not happen
            Message m = messageFactory.createInfoMessage("Invalid item ID.");
            proxy.send(userName, m);
            return;
        }// user did not declare interest
        if (!currentItem.getInterestedUsers().contains(userName)) {
            Message m = messageFactory.createInfoMessage(
                    "You have not declared interest for this item.");
            proxy.send(userName, m);
            return;
        }
         // there is already a higher bid waiting validation
        if (currentItem.getTempBid() >= amount)
            return;

        if (State.ACCEPTING_BIDS == state) {
            // temp bid is a bid that is higher than the current but
            // needs to be validated by our peer
            currentItem.setTempBid(amount);
            Message m = messageFactory.createGotBidMessage(userName, amount);
            proxy.send(Auctioneer.peerName, m);
        } else if (State.READY_TO_END == state) {
            currentItem.addPendingBid(userName, message);
            debug("saved pending bid from " + userName + ", amount = " + amount);
        } else {
            Message m = messageFactory.createInfoMessage("Not accepting bids at the moment.");
            proxy.send(userName, m);
        }
    }

    private void handleReadyToEnd()
    {
```

```
        debug("Got ready to end message");
        peerState = State.READY_TO_END;

        // if our bidTimer has already ended and there is no tempBid for which
                //we are expecting confirmation
        if ((state == State.READY_TO_END)
                    && (currentItem.getTempBid() == currentItem.getCurrentBid())) {
            currentItem.incrDiscountRound();
            int discountRound = currentItem.getDiscountRound();
            if (currentItem.getCurrentBidder().equals(Constants.no_holder)
                        && (discountRound < 5)) {
                int newPrice = (currentItem.getStartingPrice()
                                * (10 - discountRound)) / 10;
                currentItem.setCurrentBid(newPrice);
                currentItem.setTempBid(newPrice);
                Message bidAgain = messageFactory.createNewHighBidMessage(
                                                Constants.no_holder, newPrice);
                proxy.broadcast(currentItem.getInterestedUsers(), bidAgain);
                debug("sent bid again from handleReadyToEnd");
                setBidTimer();
            } else {
                anounceWinner();
                peerState = State.ACCEPTING_INTERESTS;
                beginNextAuction();
            }
        }
    }

    private void handleReadyToRun()
    {
        debug("Got ready to run message");
        peerState = State.READY_TO_BEGIN;

        if (state == State.READY_TO_BEGIN)
            beginNextAuction();
    }

    private void handleInterest(String name, Message message)
    {
        if (state == State.ACCEPTING_INTERESTS) {
            int id = Integer.parseInt(message.getProperty(Constants.item_id));
            if (currentItem.getId() == id) {
                currentItem.addUser(name);
                debug("Added user " + name +
                    " to interestedUsers list for the current item");
                Message m = messageFactory.createInfoMessage(
                        "You will now receive updates for this item.");
                proxy.send(name, m);
            } else {
                Message m = messageFactory.createInfoMessage("Invalid item ID.");
```

```
                proxy.send(name, m);
            }
        } else {
            Message m = messageFactory.createInfoMessage(
                                "Not accepting interests at the moment.");
            proxy.send(name, m);
        }
    }

    /**
     * Handle a connect request
     * @param messageProps
     * @param existingName
     * @param channel
     * @param clientProps
     */
    private void handleConnect(String existingName, Message message, SocketChannel channel,
                                    Map<String, String> clientProps)
    {
        if (existingName != null) // this user has already connected.
            return;
        String username = message.getProperty(Constants.username);
        if (proxy.isUserActive(username)) {
            Message m = messageFactory.createMessage(Constants.duplicate_name);
            proxy.send(channel, m);
            return;
        }
        proxy.addUser(username, channel);
        clientProps.put(Auctioneer.clientName, username);
        Message m = messageFactory.createInfoMessage("You are now connected to the server.");
        proxy.send(username, m);
    }

    @SuppressWarnings("unused")
    private void printConfData()
    {
        debug("Configuration for " + type + ":");
        debug("TimeLapse = " + timeLapse);
        for (Item i : items)
            debug("Item " + i.getId() + ": Starting Price = " +
                    i.getStartingPrice() + ", Description = \"" +
                    i.getDescription() + "\"");
    }

    public String getType()
    {
        return type;
    }

    public Lock getLock()
```

```
    {
        return lock;
    }

    public void debug(String debugMessage)
    {
        // if (type.equals("callee")) System.out.print(");
        debugLock.lock();
        System.out.print(type + ": ");
        DateFormat dateFormat = new SimpleDateFormat("HH:mm:ss : ");
        Date date = new Date();
        System.out.print(dateFormat.format(date));
        if (debugMessage.contains("\n"))
            System.out.print(debugMessage);
        else
            System.out.println(debugMessage);
        debugLock.unlock();
    }

    private void setState(State state)
    {
        this.state = state;
        debug("Changing State to " + state);
    }

    public boolean haveAuctionsEnded()
    {
        return auctionsEnded;
    }

    public void removeUserFromInterested(String userName)
    {
        if (currentItem != null)
            currentItem.getInterestedUsers().remove(userName);
    }
}
```

Proxy

Ο Proxy είναι όπως έχουμε ξαναπεί, η κλάση που αναλαμβάνει να υλοποιήσει την επικοινωνία. Κάθε auctioneer έχει τον Proxy του προκειμένου να μιλά με τον άλλον auctioneer, αλλά και με τους bidders. Ο κάθε auctioneer μέσω του proxy κάνει πρώτα connect και αμέσως μετά accept peer προκειμένου να συνδεθεί και ο άλλος auctioneer. Μόλις συνδεθούν και οι δύο, μπορούν πλέον να αρχίσουν να ανταλλάσσουν μηνύματα. Ο proxy φτιάχνει έναν selector που ακούει όλους τους peers και τον οποίον χρησιμοποιεί ο auctioneer. Ο proxy «ξυπνάει» είτε όταν έρθει καινούριος χρήστης (και άρα πρέπει να τον κάνει

connect) ή όταν του έρθει κάποιο μήνυμα και πρέπει να το στείλει κάπου. Έχει επιπλέον τη δυνατότητα να στέλνει είτε απευθείας το μήνυμα σε συγκεκριμένο χρήστη, ή να το στέλνει σε ολόκληρο το κανάλι.

Παρακάτω θα αναλυθούν οι μέθοδοι του proxy.

- **isUserActive**: Επιστρέφει `true` εάν το όνομα αυτό χρησιμοποιείται από κάποιον χρήστη.
- **addUser**: Μέθοδος για να προσθέτουμε κάποιον χρήστη στη λίστα του proxy με τους ενεργούς users.
- **removeUser**: Μέθοδος για να αφαιρούμε κάποιον χρήστη από τη λίστα του proxy με τους ενεργούς users.
- **connectToPeer**: Μέθοδος για να συνδεόμαστε με κάποιον auctioneer peer. Στο τέλος της μεθόδου, τυπώνεται κατάλληλο μήνυμα.
- **acceptPeer**: Μέθοδος για να δεχόμαστε μία σύνδεση από κάποιον remote peer. Στο τέλος της μεθόδου, τυπώνεται κατάλληλο μήνυμα.
- **registerPeer**: Μέθοδος προκειμένου να γίνει register ο δοσμένος selector στο κανάλι των peers.
- **broadcast**: Μέθοδος για να κάνουμε broadcast ένα μήνυμα σε όλους τους συνδεδεμένους χρήστες.
- **broadcast**: Μέθοδος για να στέλνουμε ένα μήνυμα σε όλους τους χρήστες που είναι στη λίστα των ενδιαφερόμενων.
- **send**: Μέθοδος για να στέλνουμε ένα μήνυμα σε κάποιον συγκεκριμένο χρήστη.
- **send**: Μέθοδος για να στέλνουμε ένα μήνυμα σε ένα συγκεκριμένο κανάλι.
- **shutdown**: Μέθοδος για να κλείνουμε όλο το κανάλι.
- **closeConnection**: Μέθοδος για να κλείσουμε τη σύνδεση ενός χρήστη.

Παρακάτω παρατίθεται και ο κώδικάς μας για την κλάση Proxy.

```
import java.io.IOException;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.CharBuffer;
import java.nio.channels.ClosedChannelException;
import java.nio.channels.SelectionKey;
import java.nio.channels.Selector;
import java.nio.channels.ServerSocketChannel;
import java.nio.channels.SocketChannel;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.locks.Lock;


public class Proxy
{
  private int port;
  private int peerPort;
  private Auctioneer auctioneer;
  private SocketChannel peerChannel;
  private HashMap<String, SocketChannel> activeChannels;
  private Lock lock;
  private long timeToCheck;
  private Selector selector;

  /**
   * Constructor for the proxy server
   * @param port
   * @param auctioneer
   */
  public Proxy(int port, Auctioneer auctioneer)
  {
    this.port     = port;
    this.auctioneer = auctioneer;
    this.lock     = auctioneer.getLock();
    timeToCheck   = 1000;
    activeChannels  = new HashMap<String, SocketChannel>();
  }

  /**
   * Returns true if a username is already in use
   * @param username
   * @return
   */
```

```
  public boolean isUserActive (String username)
  {
    return activeChannels.containsKey(username);
  }

  /**
   * Adds user to the proxy's list of active users
   * @param username
   * @param channel
   */
  public void addUser(String username, SocketChannel channel)
  {
    activeChannels.put(username, channel);
  }

  /**
   * Removes user from the proxy's list of active users
   * @param username
   * @param channel
   */
  public void removeUser(String username)
  {
    activeChannels.remove(username);
  }

  /**
   * Connects to an Auctioneer peer
   * @param peerPort
   * @throws IOException
   */
  private void connectToPeer(int peerPort) throws IOException
  {
    auctioneer.debug("Connecting to remote peer");
    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
//      e.printStackTrace();
    }

    peerChannel = SocketChannel.open();
    peerChannel.configureBlocking(false);
    peerChannel.connect(new InetSocketAddress("localhost", peerPort));

    while (!peerChannel.finishConnect())
      auctioneer.debug("Connecting to peer ...");

    auctioneer.debug("Connected to remote peer");
  }

  /**
```

```
   * Accepts a connection from a remote peer
   * @param channel
   * @throws IOException
   */
  private void acceptPeer(ServerSocketChannel channel) throws IOException
  {
    auctioneer.debug("Now accepting peers!");
    while (peerChannel == null) {
      peerChannel = channel.accept();
      try {
        Thread.sleep(50);
      } catch (InterruptedException e) {
//        e.printStackTrace();
      }
    }
    peerChannel.configureBlocking(false);

    auctioneer.debug("Peer accepted!");
  }

  /**
   * Register the given selector to the peer channel
   * @param selector
   * @throws ClosedChannelException
   */
  private void registerPeer(Selector selector) throws ClosedChannelException
  {
    SelectionKey peerKey = peerChannel.register(
        selector, SelectionKey.OP_READ,
        SelectionKey.OP_WRITE);

    Map<String, String> clientproperties = new HashMap<String, String>();
    clientproperties.put(Auctioneer.channelType, Auctioneer.clientChannel);
    clientproperties.put(Auctioneer.bytesReadString, "");
    clientproperties.put(Auctioneer.clientName, Auctioneer.peerName);
    peerKey.attach(clientproperties);

  }

  public void setPeerPort(int peerPort)
  {
    this.peerPort = peerPort;
  }

  /**
   * Broadcast a message to all the connected bidders
   * @param message
   */
  public void broadcast(Message message)
  {
```

```
    CharBuffer buffer;
    for (SocketChannel channel : activeChannels.values())
    {
      buffer = CharBuffer.wrap(message.toString());
      try {
        while (buffer.hasRemaining())
          channel.write(Charset.defaultCharset().encode(buffer));
      } catch (IOException e) {
//        e.printStackTrace();
      }
      buffer.clear();
    }
  }

  /**
   * Broadcast a message to all the interested bidders
   * @param message
   */
  public void broadcast(ArrayList<String> users, Message message)
  {
    for (String user : users)
      send(user, message);
  }

  /**
   * Send a message to a user
   * @param user
   * @param message
   */
  public void send(String user, Message message)
  {
    if (Auctioneer.peerName.equals(user)) {
      send(peerChannel, message);
    } else {
      SocketChannel channel = activeChannels.get(user);
      send(channel, message);
    }
  }

  /**
   * Sends the message to the specified channel
   * @param channel
   * @param message
   */
  public void send(SocketChannel channel, Message message)
  {
    CharBuffer buffer = CharBuffer.wrap(message.toString());
    try {
      while (buffer.hasRemaining())
        channel.write(Charset.defaultCharset().encode(buffer));
```

```
    } catch (IOException e) {
//      e.printStackTrace();
    }
    buffer.clear();
  }

  /**
   * Listens for incoming messages and connections
   * @throws IOException
   */
  public void run() throws IOException
  {
    auctioneer.debug("Auctioneer starting ...");

    ServerSocketChannel channel = ServerSocketChannel.open();
    channel.bind(new InetSocketAddress("localhost", port));
    channel.configureBlocking(false);
    selector = Selector.open();
    SelectionKey socketServerSelectionKey = channel.register(selector, SelectionKey.OP_ACCEPT);

    Map<String, String> properties = new HashMap<String, String>();
    properties.put(Auctioneer.channelType, Auctioneer.serverChannel);
    socketServerSelectionKey.attach(properties);

    if (auctioneer.getType().equals("caller"))
      connectToPeer(peerPort);
    else
      acceptPeer(channel);

    registerPeer(selector);


    // wait for the selected keys
    for (;;) {

      try {
        if (selector.select(timeToCheck) == 0) {
          if (auctioneer.haveAuctionsEnded()) {
            shutdown();
            break;
          }
          else {
            continue;
          }
        }
      } catch (IOException e1) {
        auctioneer.debug("Selector throwed exception: " + e1);
        continue;
      }
```

```
lock.lock();

Set<SelectionKey> selectedKeys = selector.selectedKeys();
Iterator<SelectionKey> iterator = selectedKeys.iterator();

while (iterator.hasNext()) {
  SelectionKey key = iterator.next();

  // An incoming connection
  if (((Map<?, ?>) key.attachment())
          .get(Auctioneer.channelType)
          .equals(Auctioneer.serverChannel)) {

    ServerSocketChannel serverSocketChannel = (ServerSocketChannel) key.channel();
    SocketChannel clientSocketChannel;

    try {
      clientSocketChannel = serverSocketChannel.accept();

      if (clientSocketChannel != null) {
        // set the client connection to be non blocking
        clientSocketChannel.configureBlocking(false);
        SelectionKey clientKey =
          clientSocketChannel.register(selector,
                                       SelectionKey.OP_READ,
                                       SelectionKey.OP_WRITE);

        Map<String, String> clientproperties = new HashMap<String, String>();
        clientproperties.put(Auctioneer.channelType, Auctioneer.clientChannel);
        clientproperties.put(Auctioneer.bytesReadString, "");
        clientproperties.put(Auctioneer.clientName, null);
        clientKey.attach(clientproperties);
        auctioneer.debug("Connection accepted!");
      }
    } catch (IOException e) {
      auctioneer.debug("Accepting client connection exception: " + e);
      continue;
    }
  } else {
    // data is available for read
    ByteBuffer buffer = ByteBuffer.allocate(256);
    SocketChannel clientChannel = (SocketChannel) key.channel();

    int bytesRead = 0;
    if (key.isReadable()) {
      try {
        if ((bytesRead = clientChannel.read(buffer)) > 0) {
          buffer.flip();
          String s = Charset.defaultCharset().decode(buffer).toString();
          //auctioneer.debug("Got message: " + s);
```

```
              auctioneer.processMessage(s, key);
              buffer.clear();
            }
          } catch (IOException e) {
            try {
              clientChannel.close();
            } catch (IOException e1) {
            }
            @SuppressWarnings("unchecked")
            Map<String, String> clientProps = (Map<String, String>) key.attachment();
            String userName  = clientProps.get(Auctioneer.clientName);
            if (userName != null) {
              if (userName.equals(Auctioneer.peerName)) {
                auctioneer.debug("Connection with the peer was dropped, exiting...");
                shutdown();
                break;
              } else {
                activeChannels.remove(userName);
                auctioneer.removeUserFromInterested(userName);
                auctioneer.debug("User " + userName + " disconnected.");
              }
            }
          }

          if (bytesRead < 0)
            try {
              clientChannel.close();
            } catch (IOException e) {
            }
        }
      }

      iterator.remove();
    }
    lock.unlock();
  }
}

private void shutdown()
{
  for (SocketChannel c: activeChannels.values()) {
    try {
      c.close();
    } catch (IOException e) {
//      e.printStackTrace();
    }
  }
  try {
    peerChannel.close();
  } catch (IOException e) {
```

```
//      e.printStackTrace();
    }
    try {
      selector.close();
    } catch (IOException e) {
//      e.printStackTrace();
    }
  }

  public void closeConnection(String userName)
  {
    try {
      activeChannels.remove(userName).close();
      auctioneer.debug("User \"" + userName + "\" quit.");
    } catch (IOException e) {
//      e.printStackTrace();
    }
  }
}
```

ConfParser.

Η κλάση αυτή παίρνει το αρχείο εισόδου και φτιάχνει τη λίστα με τα αντικείμενα, τις αρχικές τους τιμές και τις περιγραφές.

Παραθέτουμε απευθείας τον κώδικα της κλάσης:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;


public class ConfParser
{
    private String confFile;
    private int timeLapse;
    private ArrayList<Item> items;

    public ConfParser(String confFile)
    {
        this.confFile = confFile;
        items = new ArrayList<Item>();
    }

    public void parse() throws IOException
    {
        BufferedReader in = new BufferedReader(new FileReader(confFile));
```

```
        String line = in.readLine();
        timeLapse = Integer.parseInt(line.trim());

        line = in.readLine();
        int numItems = Integer.parseInt(line.trim());

        for (int i = 0; i < numItems; i++) {
            line = in.readLine().trim();
            int splitPos = line.indexOf(' ');
            int startingPrice = Integer.parseInt(line.substring(0, splitPos));
            String description = line.substring(splitPos + 1);
            items.add(new Item(startingPrice, description, i+1));
        }

        in.close();
    }

    public ArrayList<Item> getItems()
    {
        return items;
    }

    public int getTimeLapse()
    {
        return timeLapse;
    }

}
```

Constants.

Η κλάση αυτή κρατάει όλες τις σταθερές που χρησιμοποιεί όλος ο υπόλοιπος κώδικας. Παρακάτω παρατίθεται ο κώδικας της κλάσης:

```
public class Constants
{
        // bidder to auctioneer commands
        public static final String connect                    = "connect";
        public static final String i_am_interested  = "i_am_interested";
        public static final String my_bid                     = "my_bid";
        public static final String quit                     = "quit";

        // auctioneer to bidder commands
        public static final String start_bidding    = "start_bidding";
        public static final String new_high_bid     = "new_high_bid";
        public static final String stop_bidding     = "stop_bidding";
        public static final String duplicate_name   = "duplicate_name";
        public static final String bid_item         = "bid_item";
        public static final String auction_complete = "auction_complete";
```

```
        public static final String info             = "info";

        // peer to peer commands
        public static final String ready_to_run     = "ready_to_run";
        public static final String ready_to_end     = "ready_to_end";
        public static final String got_bid          = "got_bid";
        public static final String bid_ok           = "bid_ok";
        public static final String interested_count = "interested_count";

        // message property names
        public static final String username         = "username";
        public static final String item_id          = "item_id";
        public static final String description      = "description";
        public static final String starting_price   = "starting_price";
        public static final String winner           = "winner";
        public static final String highest_bid      = "highest_bid";
        public static final String message          = "message";
        public static final String amount           = "amount";

        // other constants
        public static final String no_holder        = "no_holder";
}
```

DBServer.

Η κλάση αυτή διαχειρίζεται τη βάση δεδομένων μας. Κάθε φορά που καλείται η αντίστοιχη μέθοδος, γίνεται κάποια ενημέρωση στη βάση δεδομένων.

Αναλυτικότερα οι μέθοδοι που χρησιμοποιήσαμε εδώ ήταν οι εξής:

- **initAuctions**: Μέθοδος για να αρχικοποιήσουμε τη βάση. Αρχικά, διαγράφουμε ό, τι υπήρχε πριν στη βάση και στη συνέχεια τοποθετούμε στη βάση όλα τα αντικείμενα, τα οποία έχουμε διαβάσει από το αρχείο.
- **updateItemPrice**: Μέθοδος για να ενημερώνουμε την τιμή του αντικειμένου.
- **execute**: Μέθοδος για να εκτελούμε κάποιο query.
- **updateBid**: Μέθοδος για να κανουμε update το bid που έχει δοθεί για κάποιο αντικείμενο.

Παρακάτω παρατίθεται και ο κώδικας της κλάσης:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;

public class DBServer
{
    private Connection conn = null;
    private Auctioneer auctioneer;

    private static DBServer callerInstance;
    private static DBServer calleeInstance;

    private DBServer() {}

    public static DBServer getCallerInstance()
    {
        if (callerInstance == null) {
            callerInstance = new DBServer();
            callerInstance.init("jdbc:mysql://localhost:3306/DB", "DB", "pass");
        }

        return callerInstance;
    }

    public static DBServer getCalleeInstance()
    {
        if (calleeInstance == null) {
            calleeInstance = new DBServer();
            calleeInstance.init("jdbc:mysql://localhost:3306/DB2", "DB", "pass");
        }

        return calleeInstance;
    }

    private void init(String dbURL, String userName, String password) {
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            conn = DriverManager.getConnection(dbURL, userName, password);
        } catch (Exception e) {
        }
    }

    public void initAuctions(ArrayList<Item> items)
    {
        String query = "Delete from items;";
        if (executeUpdate(query) == -1)
```

```
            auctioneer.debug("Unable to clear the existing items table in the database");

        for (Item i : items) {
            query = "Insert into items values(\"" +
                    i.getId() + "\", \"" +
                    i.getStartingPrice() + "\", \"" +
                    i.getDescription() + "\", null, null);";
            if (executeUpdate(query) == -1)
                auctioneer.debug("Insert Query Failed");
        }
    }

    public void updateItemPrice(int id, int bid, String bidder)
    {
        String query = "Update items set bid = \""
                    +bid+"\", bidder = \""+bidder+"\" where id = \""+id+"\";";
        if (executeUpdate(query) == -1)
            auctioneer.debug("Bid insertion into the database failed");
    }

    public ResultSet execute(String query)
    {
        auctioneer.debug("Querying: " + query);
        try {
            PreparedStatement stmt = conn.prepareStatement(query);
            return stmt.executeQuery();
        } catch (Exception e) {
            return null;
        }
    }

    public int executeUpdate(String query)
    {
        auctioneer.debug("Querying: " + query);
        try {
            PreparedStatement stmt = conn.prepareStatement(query);
            return stmt.executeUpdate();
        } catch (Exception e) {
            return -1;
        }
    }

    public void setAuctioneer(Auctioneer auctioneer)
    {
        this.auctioneer = auctioneer;
    }

    public void updateBid(Item item)
    {
        String query = "Update items set bid = \""
```

```
                        +item.getCurrentBid()+"\", bidder = " +
                    "\""+item.getCurrentBidder()+"\" where id = \""
                                    +item.getId()+"\";";
        if (executeUpdate(query) == -1)
            auctioneer.debug("Bid insertion into the database failed");
    }

}
```

Item.

Η κλάση Item είναι η κλάση του Server που χειρίζεται τα αντικείμενα που παίζονται στο auction. Συγκεκριμένα, κρατάει πληροφορίες σχετικά με το id, την αρχική τους τιμή, την περιγραφή, μία λίστα με τους ενδιαφερόμενους χρήστες, το tempBid, το currentBid καθώς και το discountRound στο οποίο βρισκόμαστε.

Παραθέτουμε απευθείας τον κώδικα της κλάσης Item

```
import java.util.ArrayList;
import java.util.HashMap;


public class Item
{
   private int id;
   private int startingPrice;
   private String description;
   private ArrayList<String> interestedUsers;
   private String currentBidder;
   private int currentBid;
   private int tempBid;
   private HashMap<String, Message> pendingBids;
   private int peerInterestedCount;
   private int discountRound;

   public Item(int startingPrice, String description, int id)
   {
      this.id              = id;
      this.startingPrice   = startingPrice;
      this.description     = description;
      this.interestedUsers = new ArrayList<String>();
      this.pendingBids     = new HashMap<String, Message>();
      this.currentBidder   = Constants.no_holder;
      this.currentBid      = startingPrice;
      this.tempBid         = startingPrice;
   }

   public void addUser(String username)
```

```
    {
        this.interestedUsers.add(username);
    }

    public int getId()
    {
        return id;
    }

    public int getStartingPrice() {
        return startingPrice;
    }

    public String getDescription() {
        return description;
    }

    public ArrayList<String> getInterestedUsers()
    {
        return interestedUsers;
    }

    public String getCurrentBidder()
    {
        return currentBidder;
    }

    public int getCurrentBid()
    {
        return currentBid;
    }

    public void setCurrentBidder(String currentBidder)
    {
        this.currentBidder = currentBidder;
    }

    public void setCurrentBid(int currentBid)
    {
        this.currentBid = currentBid;
    }

    public HashMap<String, Message> getPendingBids()
    {
        return pendingBids;
    }

    public void addPendingBid(String userName, Message pendingBid)
    {
        pendingBids.put(userName, pendingBid);
```

```
    }

    public void clearPendingBids()
    {
        pendingBids.clear();
    }

    public int getTempBid()
    {
        return tempBid;
    }

    public void setTempBid(int tempBid)
    {
        this.tempBid = tempBid;
    }

    public int getPeerInterestedCount()
    {
        return peerInterestedCount;
    }

    public void setPeerInterestedCount(int peerInterestedCount)
    {
        this.peerInterestedCount = peerInterestedCount;
    }

    public int getDiscountRound()
    {
        return discountRound;
    }

    public void incrDiscountRound()
    {
        this.discountRound++;
    }
}
```

Message.

Η κλάση Message είναι κλάση που χρησιμοποιούν τόσο οι Auctioneers, όσο και οι Bidders. Συγκεκριμένα, ό, τι μήνυμα ανταλλάσουν μεταξύ τους οι servers, αλλά και οι bidders με τους servers είναι αντικείμενο αυτής της κλάσης. Οι μέθοδοι αυτής της κλάσης χρησιμεύουν ώστε όταν λαμβάνουμε ένα μήνυμα, να μπορούμε να εξάγουμε τα properties του μηνύματος, τα οποία θα χρησιμοποιηθούν στη συνέχεια από τη messageFactory.

Παρακάτω παρατίθεται ο κώδικας της κλάσης Message

```
import java.util.HashMap;


public class Message
{
    private String command;
    private HashMap<String, String> properties;

    public Message(String command)
    {
        this.command = command;
        properties = new HashMap<String, String>();
    }

    public void addProperty(String name, String value)
    {
        properties.put(name, value);
    }

    public void addProperty(String name, int value)
    {
        properties.put(name, String.format("%d", value));
    }

    public HashMap<String, String> getProperties()
    {
        return properties;
    }

    public String getProperty(String name)
    {
        return properties.get(name);
    }

    public String toString()
    {
        String message = command + ": ";
        for (String key : properties.keySet())
            message += key + " = \"" + properties.get(key) + "\", ";

        if (properties.isEmpty())
            return message + "\n";
        else
            return message.substring(0, message.length() - 2) + "\n";
    }

    public String getCommand()
    {
```

```
        return command;
    }
}
```

MessageFactory.

Η κλάση αυτή είναι υπεύθυνη για τον χειρισμό των μηνυμάτων, τη δημιουργία αυτών (αν μας δοθούν τα κατάλληλα properties), αλλά και την εξαγωγή πληροφοριών από τα μηνύματα που δεχόμαστε. Συγκεκριμένα, έχουμε τις εξής μεθόδους:

- **createMessage**: Μέθοδος για τη δημιουργία ενός μηνύματος χωρίς καθόλου properties.
- **createMessage**: Μέθοδος για τη δημιουργία ενός μηνύματος με τις δοσμένες properties. Οι properties μπαίνουν σε ένα HashMap για να μπορούμε να έχουμε γρήγορη και εύκολη πρόσβαση σε αυτές.
- **createBidItemMessage**: Μέθοδος που δέχεται το id του αντικειμένου, την περιγραφή του και μια αρχική τιμή και δημιουργεί ένα μήνυμα με τις κατάλληλες ιδιότητες.
- **createStartBiddingMessage**: Μέθοδος που δέχεται το id του αντικειμένου, και μια αρχική τιμή και δημιουργεί ένα μήνυμα με τις κατάλληλες ιδιότητες.
- **createStopBiddingMessage**: Μέθοδος που δέχεται το id του αντικειμένου, το όνομα του νικητή και το highestBid και δημιουργεί ένα μήνυμα με τις κατάλληλες ιδιότητες.
- **createQuitMessage**: Μέθοδος που δέχεται το username ενός χρήστη και δημιουργεί ένα μήνυμα με τις κατάλληλες ιδιότητες.
- **createIAmInterestedMessage**: Μέθοδος που δέχεται το username ενός χρήστη και το id ενός αντικειμένου και δημιουργεί ένα μήνυμα με τις κατάλληλες ιδιότητες.
- **createInterestedCountMessage**: Μέθοδος που δέχεται τον αριθμό των ενδιαφερόμενων χρηστών και δημιουργεί ένα μήνυμα με τις κατάλληλες ιδιότητες.
- **createGotBidMessage**: Μέθοδος που δέχεται το username ενός χρήστη και το ποσό του bid και δημιουργεί ένα μήνυμα με τις κατάλληλες ιδιότητες.
- **createBidOkMessage**: Μέθοδος που δέχεται το username ενός χρήστη και το ποσό του bid και δημιουργεί ένα μήνυμα με τις κατάλληλες ιδιότητες.

- **createNewHighBidMessage**: Μέθοδος που δέχεται το username ενός χρήστη και το ποσό του bid και δημιουργεί ένα μήνυμα με τις κατάλληλες ιδιότητες.
- **fromRawMessage**: Μέθοδος για τη δημιουργία ενός message object από το string που μας έχει δοθεί.
- **getProperties**: Μέθοδος που δέχεται ένα string και επιστρέφει ένα HashMap με τα αντίστοιχα properties και τις τιμές τους, που προκύπτουν από το string.
- **getCommand**: Μέθοδος που επιστρέφει την εντολή, από ένα δοσμένο μήνυμα.
- **extractMessages**: Μέθοδος που εξάγει τα ολοκληρωμένα μηνύματα που περιέχονται στη δοσμένη είσοδο. Αποθηκεύει προσωρινά τα data που απομένουν, τα οποία θα τα χειριστεί η μέθοδος getRemaining.
- **extractRawMessages**: Μέθοδος για να σπάσουμε ένα string που μας έχει δοθεί σε ξεχωριστά μηνύματα. Αποθηκεύει προσωρινά τα data που απομένουν, τα οποία θα τα χειριστεί η μέθοδος getRemaining.

Παρακάτω παρατίθεται και ο κώδικας της κλάσης MesageFactory.

```
import java.util.ArrayList;
import java.util.HashMap;

public class MessageFactory
{
    private String remaining;

    /**
     * Creates a message object with no properties
     * @param command
     * @return
     */
    public Message createMessage(String command)
    {
        return new Message(command);
    }

    /**
     * Creates a message object with the given properties
     * @param command
     * @param properties
     * @return
     */
```

```
public Message createMessage(String command, HashMap<String, String> properties)
{
    Message message = new Message(command);
    for (String key : properties.keySet())
        message.addProperty(key, properties.get(key));

    return message;
}

public Message createBidItemMessage(int itemId, String description, int startingPrice)
{
    Message message = createMessage(Constants.bid_item);
    message.addProperty(Constants.item_id, itemId);
    message.addProperty(Constants.description, description);
    message.addProperty(Constants.starting_price, startingPrice);
    return message;
}

public Message createStopBiddingMessage(int itemId, String winner, int highestBid)
{
    Message message = createMessage(Constants.stop_bidding);
    message.addProperty(Constants.item_id, itemId);
    message.addProperty(Constants.winner, winner);
    message.addProperty(Constants.highest_bid, highestBid);
    return message;
}

public Message createStartBiddingMessage(int startingPrice, int itemId)
{
    Message message = createMessage(Constants.start_bidding);
    message.addProperty(Constants.starting_price, startingPrice);
    message.addProperty(Constants.item_id, itemId);
    return message;
}
/**
 * Creates a message for quitting a user and shutting his/her channel down
 * @param username
 */

public Message createQuitMessage(String username)
{
    Message message = createMessage(Constants.quit);
    message.addProperty(Constants.username, username);
    return message;
}

/**
 * Creates a message for a control message: i_am_interested
 * @param username
 * @param id
```

```
     */

    public Message createIAmInterestedMessage(String username, int id)
    {
        Message message = createMessage(Constants.i_am_interested);
        message.addProperty(Constants.username, username);
        message.addProperty(Constants.item_id, id);
        return message;
    }

    /**
     * Creates a message for placing a new bid on behalf of a user
     * @param username
     * @param amount
     */
    public Message createMyBidMessage(String username, int amount, int itemId)
    {
        Message message = createMessage(Constants.my_bid);
        message.addProperty(Constants.username, username);
        message.addProperty(Constants.amount, amount);
        message.addProperty(Constants.item_id, itemId);
        return message;
    }

    public Message createInterestedCountMessage(int count)
    {
        Message message = createMessage(Constants.interested_count);
        message.addProperty(Constants.amount, count);
        return message;
    }

    public Message createInfoMessage(String message)
    {
        Message m = createMessage(Constants.info);
        m.addProperty(Constants.message, message);
        return m;
    }

    public Message createGotBidMessage(String userName, int amount)
    {
        Message message = createMessage(Constants.got_bid);
        message.addProperty(Constants.username, userName);
        message.addProperty(Constants.amount, amount);
        return message;
    }

    public Message createBidOkMessage(String userName, int amount)
    {
        Message message = createMessage(Constants.bid_ok);
        message.addProperty(Constants.username, userName);
```

```
        message.addProperty(Constants.amount, amount);
        return message;
    }

    public Message createNewHighBidMessage(String userName, int amount)
    {
        Message message = createMessage(Constants.new_high_bid);
        message.addProperty(Constants.username, userName);
        message.addProperty(Constants.amount, amount);
        return message;
    }

    /**
     * Creates a message object from its String representation
     * @param rawMessage
     * @return
     */
    public Message fromRawMessage(String rawMessage)
    {
        String command = getCommand(rawMessage);
        HashMap<String, String> props = getProperties(rawMessage);

        return createMessage(command, props);
    }

    /**
     * Returns a HashMap containing all the arguments and their values found in the message
     * @param message
     * @return
     */
    private HashMap<String, String> getProperties(String message)
    {
        int pos = message.indexOf(": ");
        String props = message.substring(pos + 2);
        HashMap<String, String> properties = new HashMap<String, String>();

        for (String prop : props.split(", ")) {
            String[] pair = prop.split(" = ");
            if (pair.length == 2)
                properties.put(pair[0], pair[1].replace("\"", ""));
        }
        return properties;
    }

    /**
     * Returns the command of the given message
     * @param message
     * @return
     */
    private String getCommand(String message)
```

```
    {
        int pos = message.indexOf(": ");
        return message.substring(0, pos);
    }

    /**
     * Extracts the complete messages contained in the given data
     * Stores temporarily the remaining data to be retrieved through getRemaining()
     * @param data
     * @return
     */
    public ArrayList<Message> extractMessages(String data)
    {
        ArrayList<String> rawMessages = extractRawMessages(data);
        ArrayList<Message> messages = new ArrayList<Message>();
        for (String m : rawMessages)
            messages.add(fromRawMessage(m));
        return messages;
    }

    /**
     * Breaks a string of received data into separate messages
     * Stores temporarily the remaining data to be retrieved through getRemaining()
     * @param data
     * @return
     */
    private ArrayList<String> extractRawMessages(String data)
    {
        ArrayList<String> list = new ArrayList<String>();
        int pos;
        while ((pos = data.indexOf('\n')) != -1) {
            String message = data.substring(0, pos);
            data = data.substring(pos+1);
            list.add(message);
        }

        this.remaining = data;
        return list;
    }

    /**
     * Returns the remaining data left over from the last extractMessages call
     * @return
     */
    public String getRemaining()
    {
        return remaining;
    }
}
```

3.1.2 Κώδικας των Clients

ClientLauncher

Το ClientLauncher είναι το πρόγραμμα το οποίο τρέχουμε για να αρχικοποιήσουμε τους Clients. Παίρνει ως είσοδο τρία arguments: το πρώτο είναι ο host που θα τρέξει ο bidder, το δεύτερο, η port που θα τρέξει και το τρίτο, το όνομα του bidder. Η κλάση αυτή ανοίγει ένα κανάλι και ζητάει από τον server να τον συνδέσει. Επιπλέον, αρχίζει τα δύο threads του client: ένα listeningThread που χρησιμοποιείται για να ακούμε από το κανάλι αυτά που μας στέλνει ο server και ένα commandThread που χρησιμοποιείται για να προωθούμε τις εντολές του χρήστη στον server.

Παρακάτω παρατίθεται ο κώδικας της κλάσης:

```
import java.io.BufferedReader;
import java.util.*;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.CharBuffer;
import java.nio.channels.SocketChannel;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;


public class ClientLauncher {
        private static int port;
        private static String host;
        private static SocketChannel channel;
        public static String bidderName;

        public static void main(String[] args) throws IOException, InterruptedException
        {
                Thread.sleep(5000);
                if (args.length != 3) {
                        System.err.println("Wrong number of parameters.");
                        System.err.println("Usage: ClientLauncher <host> <port> <bidderName>");
                        System.exit(1);
                }

                host = args[0];

                try {
                        port = Integer.parseInt(args[1]);
```

```
                } catch (NumberFormatException e) {
                        System.err.println("Argument <port> must be an integer.");
                System.exit(1);
                }


                channel = SocketChannel.open();

                bidderName = args[2] + "@" +host + ":" +port;
                channel.configureBlocking(false);
                channel.connect(new InetSocketAddress(host, port));

                while (!channel.finishConnect())
                        ;



                CharBuffer buffer =
            CharBuffer.wrap("connect: username = \""+bidderName+"\"\n");

        while (buffer.hasRemaining()) {
            channel.write(Charset.defaultCharset().encode(buffer));
        }


        Client client = new Client(host, port, bidderName, channel);
        new Thread(client.listeningThread).start();
        new Thread(client.commandThread).start();
        }
}
```

Client

Η κλάση Client υλοποιεί την λειτουργικότητα του bidder. Τρέχει δυο παράλληλα threads: το commandThread ακούει το input απο τον χρήστη και αναλόγως του δίνει πληροφορίες για το τρέχον item (αν ο χρήστης είναι στη λίστα των ενδιαφερόμενων χρηστών για το αντικείμενο), τον αποσυνδέει από το κανάλι ή προωθεί το αίτημά του στον server (πχ, αν ο χρήστης ζητήσει να προστεθεί στη λίστα με τους ενδιαφερόμενους χρήστες πατώντας «i_am_interested». Το listeningThread ακούει το κανάλι για τα μηνύματα που στέλνει ο server και ανάλογα με το control message που λαμβάνει κάνει τις κατάλληλες ενέργειες.

Παρακάτω παρατίθεται ο κώδικας της κλάσης:

```
import java.io.BufferedReader;
import java.util.*;
import java.io.IOException;
import java.io.InputStreamReader;
```

```
import java.net.InetSocketAddress;
import java.nio.ByteBuffer;
import java.nio.CharBuffer;
import java.nio.channels.SocketChannel;
import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;


public class Client {
    private static int port;
    private static String host;
    private static SocketChannel channel;
    public Runnable listeningThread;
    public Runnable commandThread;
    ClientItem currentItem;
    private static String bidderName;
    private static ArrayList<ClientItem> boughtItems;
    private MessageFactory messageFactory;
    private static String total;
    private Lock itemLock;

    public Client(String host, int port, String bidderName, SocketChannel channel) {
        this.host = host;
        this.port = port;
        this.bidderName = bidderName;
        this.channel = channel;


        boughtItems = new ArrayList<ClientItem>();
        total = "";
        messageFactory  = new MessageFactory();
        itemLock = new ReentrantLock();
        listeningThread = new Runnable() {
            public void run() {
                try {
                    Client.this.runListeningThread();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        };

        commandThread = new Runnable() {
            public void run() {
                try {
                    Client.this.runCommandThread();
                } catch (IOException e) {
```

```
                e.printStackTrace();
            }
        }
    };
}

protected void runCommandThread() throws IOException {
    // TODO Auto-generated method stub

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    while (true) {
        String input;
        while((input=br.readLine())!=null) {
            System.out.println("User said: " + input);
            if (input.equals("list_high_bid")) {
                itemLock.lock();
                if (currentItem != null)
                    System.out.println("Currently highest bid: "
                                       +currentItem.getCurrentPrice());
                else
                    System.out.println("You are not in the interested users " +
                                       "list for the currentItem.");
                itemLock.unlock();
            }
            else if (input.equals("list_description")) {
                itemLock.lock();
                if (currentItem != null)
                    System.out.println("Description of item currently auctioned: "
                                       + currentItem.getDescription());
                else
                    System.out.println("You are not in the interested users " +
                                       "list for the currentItem.");
                itemLock.unlock();
            }
            else if (input.equals("quit")) {
                System.out.println("Bye bye " +bidderName);
                send(channel, messageFactory.createQuitMessage(bidderName));
                channel.close();
                /* after closing the channel, we print the items that the user bought */
                if (boughtItems != null) {
                    Iterator<ClientItem> it = boughtItems.iterator();
                    System.out.println("You bought: " + boughtItems.size() + "items!");
                    while (it.hasNext()) {
                        ClientItem bought = it.next();
                        System.out.println("Item ID = " + bought.getId());
                        System.out.println("Item Description = "+bought.getDescription());
                    }
                }
                else
```

```
                    System.out.println("You bought no items.");
                System.exit(0);
            }
            else if (input.equals("i_am_interested")) {
                itemLock.lock();
                send(channel,
                     messageFactory.createIAmInterestedMessage(bidderName,
                                                               currentItem.getId()));
                itemLock.unlock();
            }
            else {
                itemLock.lock();
                String[] temp = input.split(" ");
                if (temp[0].equals("bid") && (temp.length == 2)) {
                    System.out.println("You gave a new bid.");
                    int amount = Integer.parseInt(temp[1]);
                    if (amount > currentItem.getCurrentPrice()) {
                    //    System.out.println("Your bid got accepted.");
                        send(channel,
                             messageFactory.createMyBidMessage(bidderName,
                                                               amount,
                                                               currentItem.getId()));
                    }
                    else {
                        System.out.println("Sorry we cannot accept bids less than "
                                           + currentItem.getCurrentPrice());
                    }
                    itemLock.unlock();
                }
                else
                    System.err.println("Wrong type of input. Please try again.");
            }
        }
    }

}


// Listens on the socket channel and prints out whatever the server delivers.
protected void runListeningThread() throws IOException {


    while (true) {
        // see if any message has been received
        ByteBuffer bufferA = ByteBuffer.allocate(256);
        String message = "";
        while ((channel.read(bufferA)) > 0) {
            // flip the buffer to start reading
            bufferA.flip();
            message += Charset.defaultCharset().decode(bufferA);
```

```
            }

            if (message.length() > 0) {
                //System.out.print(message);
                processMessage(message);
                //bufferA.clear();
                //handleMessage(message);
                //message = "";
            }
        }

    }
    /**
     * Sends the message to the specified channel
     * @param channel
     * @param message
     */
    protected void send(SocketChannel channel, Message message)
    {
        CharBuffer buffer = CharBuffer.wrap(message.toString());
        try {
            while (buffer.hasRemaining())
                channel.write(Charset.defaultCharset().encode(buffer));
        } catch (IOException e) {
//            e.printStackTrace();
        }
        buffer.clear();
    }

    public void processMessage(String s) {
        //@SuppressWarnings("unchecked")

        total += s;
        ArrayList<Message> messages = messageFactory.extractMessages(total);
        String remaining = messageFactory.getRemaining();
        total = remaining;

        for (Message m : messages)
            handleMessage(m);
    }

    // General method for handling the commands that server gives
    private void handleMessage(Message m) {
        String command = m.getCommand();
        HashMap<String, String> props = m.getProperties();
        System.out.print("Got message: " +m);
        switch (command) {
        case Constants.duplicate_name:
            handleDuplicateName();
            break;
```

```
        case Constants.auction_complete:
              handleAuctionComplete();
              break;
        case Constants.start_bidding:
              itemLock.lock();
            handleStartBidding(props);
            itemLock.unlock();
            break;
        case Constants.bid_item:
            itemLock.lock();
            handleBidItem(props);
            itemLock.unlock();
            break;
        case Constants.new_high_bid:
            itemLock.lock();
            handleNewHighBid(props);
            itemLock.unlock();
            break;
        case Constants.stop_bidding:
            itemLock.lock();
            handleStopBidding(props);
            itemLock.unlock();
            break;
        case Constants.info:
            handleInfo(props);
            break;
        }
    }


    // Inform users about stop bidding
    // If client is the winner, inform him & add the item bought into his list!
    private void handleStopBidding(HashMap<String, String> props) {
        System.out.println("You shall now stop bidding for the item.");
        String winner = props.get(Constants.winner);
        if (bidderName.equals(winner)) {
            System.out.println("Congratulations! The item: " + currentItem.getDescription()
                               + " is yours!");
            boughtItems.add(currentItem);
            currentItem = null;
        }

    }

    // When you get info via the channel, just print it to the user
    private void handleInfo(HashMap<String, String> props) {
        for (String key : props.keySet())
            if (key.equals("message"))
                System.out.print(props.get(key));
        System.out.println();
```

```
    }


    // Inform users about new highest bid & holder
    private void handleNewHighBid(HashMap<String, String> props) {
        System.out.println("New highest bid for current item belongs to "
                           + props.get(Constants.username));
        System.out.println("New price: " + props.get(Constants.amount));
        currentItem.setHolder(props.get(Constants.username));
        currentItem.setCurrentPrice(Integer.parseInt(props.get(Constants.amount)));
    }

    // Inform users about the info of the new item + create the new current Item
    private void handleBidItem(HashMap<String, String> props) {
        System.out.println("New item out for bidding!");
        System.out.println("Item ID: " + props.get(Constants.item_id));
        System.out.println("Item Description: " + props.get(Constants.description));
        System.out.println("Starting Price: " +props.get(Constants.starting_price));
        int initial = Integer.parseInt(props.get(Constants.starting_price));
        int id      = Integer.parseInt(props.get(Constants.item_id));

        // create the new current Item with all of its details
        currentItem = new ClientItem(initial, props.get(Constants.description), id,
                                                    Constants.no_holder);
    }

    private void handleStartBidding(HashMap<String, String> props) {
        String temp  = props.get("item_id");
        String descr = currentItem.getDescription();
        System.out.println("You may now start bidding for item with description = "+descr+"!");
        //currentItem.id = Integer.parseInt(temp);
    }

    // Inform users about the completion of the auction
    private void handleAuctionComplete() {
        System.out.println("Auction is now completed.");
        System.out.println("Thank you for your participation");
        try {
            channel.close();
        } catch (IOException e) {
        }
        System.exit(0);
    }

    private void handleDuplicateName() {
        System.err.println("Username " + bidderName + " already exists.");
        System.exit(1);
    }

    private void printMessage(String command, HashMap<String, String> props) {
```

```
        System.out.print("Received: " + command + ": ");
        for (String key : props.keySet())
            System.out.print(key + " = \"" + props.get(key) + "\" ");
        System.out.println();
    }
}
```

ClientItem

Η κλάση αυτή χειρίζεται τα αντικείμενα που παίζουν στο auction από τη μεριά του client. Κρατά και διαχειρίζεται πολύ λιγότερες πληροφορίες από την αντίστοιχη κλάση του server, οι οποίες είναι: το id, η περιγραφή, η αρχική τιμή και ο τρέχων holder του αντικειμένου.

Παρακάτω παρατίθεται ο κώδικας της κλάσης:

```
public class ClientItem
{
    private int id;
    private int startingPrice;
    private String description;
    private String holder;

    private int currentPrice;

    public ClientItem(int startingPrice, String description, int id, String holder)
    {
        this.id = id;
        this.startingPrice = startingPrice;
        this.description = description;
        this.holder = holder;
        this.currentPrice = startingPrice;
    }

    public int getId()
    {
        return id;
    }

    public void setId(int id)
    {
        this.id = id;
    }

    public int getStartingPrice() {
        return startingPrice;
    }
```

```
    public void setStartingPrice(int starting_price) {
        this.startingPrice = starting_price;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String descr) {
        this.description = descr;
    }

    public String getHolder() {
        return holder;
    }

    public void setHolder(String hold) {
        this.holder = hold;
    }

    public int getCurrentPrice() {
        return currentPrice;
    }

    public void setCurrentPrice(int curr_price) {
        this.currentPrice = curr_price;
    }
}
```

3.2 Design Patterns

Στον κώδικά μας χρησιμοποιήσαμε 3 design patterns και συγκεκριμένα τα εξής:

- **Singleton Design Pattern.** Χρησιμοποιήσαμε το singleton design pattern για την υλοποίηση των servers. Το pattern αυτό είναι ένα από τα πιο απλά patterns στη Java. Είναι creational pattern, εφόσον προσφέρει έναν από τους καλύτερους τρόπους για να δημιουργήσεις ένα αντικείμενο.

  Το pattern αυτό περιλαμβάνει μία κλάση, η οποία είναι υπεύθυνη για τη δημιουργία ενός αντικειμένου. Η κλάση είναι επίσης υπεύθυνη για το ότι θα δημιουργηθεί μόνο ένα αντικείμενο. Προσφέρει επίσης, έναν τρόπο να έχουμε πρόσβαση σε ένα αντικείμενο άμεσα, χωρίς να χρειάζεται να αρχικοποιήσουμε το αντικείμενο της κλάσης.

- **Proxy Design Pattern.** Το proxy design pattern χρησιμοποιήθηκε για την επικοι-

νωνία μεταξύ των δύο servers με τους clients. Στο pattern αυτό ουσιαστικά, μια κλάση υλοποιεί την λειτουργικότητα μιας άλλης κλάσης. Έτσι και σε εμάς, ο proxy αναλαμβάνει να κάνει όλη την επικοινωνία.

- **Factory Design Pattern.** Το factory pattern χρησιμοποιήθηκε για τη μοντελοποίηση των μηνυμάτων που θα έστελναν μεταξύ τους servers και bidders. Μέσω αυτού του μορφήματος δημιουργούσαμε και εξάγαμε τα μηνύματα που ανταλλάσσονταν. Επιλέξαμε αυτό το pattern γιατί μας δίνει τη δυνατότητα να δημιουργήσουμε αντικείμενα, χωρίς να εκθέτουμε τη λογική της δημιουργίας στον client.

4 Driver

4.1 Ο κώδικας

Σε αυτή την ενότητα θα παραθέσουμε αρχικά τον κώδικα του driver που τρέξαμε, καθώς και τα logs απο κάποιες ενδεικτικές εκτελέσεις.

Ο driver δέχεται για όρισμα ένα αρχείο, το οποίο στην πρώτη γραμμή έχει το πλήθος των bidders που θα συνδεθούν, και στις n επόμενες, έχει τα ονόματα αυτών των χρηστών, καθώς και τις πόρτες που θα συνδεθούν. Στη συνέχεια, έχει την ακόλουθη μορφή: `<time> <bidder> <command>` Με βάση αυτή τη μορφή, τη στιγμή `<time>`, θα έχει πληκτρολογήσει ο `<bidder>`, το μήνυμα `<command>`. Μέσω του driver, κάνουμε parse τα μηνύματα και ανοίγουμε τους clients ως subprocesses. Κάθε subprocess γράφει σε ένα αρχείο το `stdout` (με κατάληξη `.out`) και ένα αρχείο με κατάληξη `.err` το `stderr`.

Παρακάτω παρατίθεται ο κώδικας του driver

```
#!/usr/bin/python

import subprocess
import os
import sys
import time

# reads from file descriptor, fd, and creates a list with triples
# (<time>, <bidderName>, <command>)
def parse_messages(fd):
  messages = []
  for i in fd:
    ln = i.strip().split()
    messages.append((int(ln[0]), ln[1], " ".join(ln[2:])))
  return messages

def parse_ports(fd):
  num_clients = int(fd.readline())
  ports = {}
  for i in range(num_clients):
    ln = fd.readline().split()
    ports[ln[0]] = ln[1]
  fd.readline()
  return ports

if len(sys.argv) != 2:
  print "Usage: " + sys.argv[0] + " [testcase]"
  exit()
```

```
# read the messages
f = open(sys.argv[1],'r')
ports = parse_ports(f)
msgList = parse_messages(f)
f.close()

# creates a subprocess for the servers
with open ("logs/server.log", 'w') as fd:
  with open ("logs/server.err", 'w') as fd2:
    subprocess.Popen(["java", "ServerLauncher", "../auct_conf.txt"], stdout = fd, stderr = fd2)

# sleeps so as to wait for the 2 peers to connect
time.sleep(1)

# send the commands to the bidders
bidder = {}
last_time = 0
for i in msgList:
  time.sleep((i[0]-last_time)/1000.0)
  last_time = i[0]
  if i[2] == "launch":
    with open ("logs/{0}@localhost:{1}.out".format(i[1], ports[i[1]]), 'w') as fd:
      with open ("logs/{0}@localhost:{1}.err".format(i[1], ports[i[1]]), 'w') as fd2:
        bidder[i[1]] = subprocess.Popen(
                        ["java", "ClientLauncher", "localhost", ports[i[1]], i[1]],
                        stdin = subprocess.PIPE,
                        stdout = fd,
                        stderr = fd2
                       )
  else:
     bidder[i[1]].stdin.write(i[2] + "\n")

# wait for all the bidders to exit
for b in bidder.itervalues():
  b.wait()
```

4.2 Testcases

Testcase1:

```
5
chara   4444
bill    5556
kostas  4444
maria   5556
giannis 4444

1000  chara   launch
1500  bill    launch
14500 chara   i_am_interested
15000 kostas  launch
30000 chara   bid 150
30500 maria   launch
31500 giannis launch
32000 kostas  list_high_bid
33000 maria   i_am_interested
34000 giannis list_description
35000 maria   bid 560
36000 bill    i_am_interested
```

Testcase1 description:

```
chara: gets connected. Sends i_am_interested for the first item.
Doesn't bid on the first round, so the item gets discounted.
Then she bids and wins the item.

Item 2 doesn't have anyone interested. Item gets discarded due to lack of interest.
```

Testcase1 server log file:

```
caller: 01:02:01 : Querying: Delete from items;
caller: 01:02:01 : Querying: Insert into items values("1", "50", "This is the first item",
null, null);
caller: 01:02:02 : Querying: Insert into items values("2", "40", "This is the second item",
null, null);
caller: 01:02:02 : Querying: Insert into items values("3", "100", "This is the last item",
null, null);
callee: 01:02:02 : Querying: Delete from items;
callee: 01:02:02 : Querying: Insert into items values("1", "50", "This is the first item",
null, null);
callee: 01:02:02 : Querying: Insert into items values("2", "40", "This is the second item",
null, null);
callee: 01:02:02 : Querying: Insert into items values("3", "100", "This is the last item",
null, null);
caller: 01:02:02 : Auctioneer starting ...
```

```
callee: 01:02:02 : Auctioneer starting ...
caller: 01:02:02 : Connecting to remote peer
callee: 01:02:02 : Now accepting peers!
caller: 01:02:03 : Connected to remote peer
caller: 01:02:03 : Connection accepted!
caller: 01:02:03 : Received: connect: username = "chara@localhost:4444"
callee: 01:02:03 : Peer accepted!
callee: 01:02:04 : Connection accepted!
callee: 01:02:04 : Received: connect: username = "bill@localhost:5556"
caller: 01:02:07 : Auctions start in 5 seconds!
caller: 01:02:12 : Auctions are beggining!
callee: 01:02:12 : Auctions are beggining!
caller: 01:02:12 : Changing State to READY_TO_BEGIN
callee: 01:02:12 : Changing State to READY_TO_BEGIN
caller: 01:02:12 : Received: ready_to_run:
callee: 01:02:12 : Received: ready_to_run:
callee: 01:02:12 : Got ready to run message
caller: 01:02:12 : Got ready to run message
caller: 01:02:12 : Starting Auction for next item
callee: 01:02:12 : Starting Auction for next item
caller: 01:02:12 : Changing State to ACCEPTING_INTERESTS
callee: 01:02:12 : Changing State to ACCEPTING_INTERESTS
caller: 01:02:16 : Received: i_am_interested: username = "chara@localhost:4444", item_id = "1"
caller: 01:02:16 : Added user chara@localhost:4444 to interestedUsers list for the current item
caller: 01:02:17 : Connection accepted!
caller: 01:02:17 : Received: connect: username = "kostas@localhost:4444"
caller: 01:02:22 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:02:22 : interest time ended!
callee: 01:02:22 : Changing State to WAITING_INTERESTED_COUNT
callee: 01:02:22 : interest time ended!
caller: 01:02:22 : Received: interested_count: amount = "0"
caller: 01:02:22 : Changing State to ACCEPTING_BIDS
callee: 01:02:22 : Received: interested_count: amount = "1"
callee: 01:02:22 : Changing State to ACCEPTING_BIDS
caller: 01:02:32 : Received: my_bid: amount = "150", username = "chara@localhost:4444",
item_id = "1"
callee: 01:02:32 : Received: got_bid: amount = "150", username = "chara@localhost:4444"
callee: 01:02:32 : New highest bid from user chara@localhost:4444, amount = 150
callee: 01:02:32 : Changing State to ACCEPTING_BIDS
caller: 01:02:32 : Received: bid_ok: amount = "150", username = "chara@localhost:4444"
caller: 01:02:32 : New highest bid from user chara@localhost:4444, amount = 150
caller: 01:02:32 : Changing State to ACCEPTING_BIDS
callee: 01:02:33 : Connection accepted!
callee: 01:02:33 : Received: connect: username = "maria@localhost:5556"
caller: 01:02:34 : Connection accepted!
caller: 01:02:34 : Received: connect: username = "giannis@localhost:4444"
callee: 01:02:38 : Received: i_am_interested: username = "bill@localhost:5556", item_id = "1"
callee: 01:02:42 : bid timer ended
callee: 01:02:42 : Changing State to READY_TO_END
caller: 01:02:42 : Received: ready_to_end:
```

```
caller: 01:02:42 : Got ready to end message
caller: 01:02:42 : bid timer ended
caller: 01:02:42 : Changing State to READY_TO_END
caller: 01:02:42 : Querying: Update items set bid = "150", bidder = "chara@localhost:4444"
where id = "1";
callee: 01:02:42 : Received: ready_to_end:
callee: 01:02:42 : Got ready to end message
callee: 01:02:42 : Querying: Update items set bid = "150", bidder = "chara@localhost:4444"
where id = "1";
caller: 01:02:42 : Starting Auction for next item
caller: 01:02:42 : Changing State to ACCEPTING_INTERESTS
callee: 01:02:42 : Starting Auction for next item
callee: 01:02:42 : Changing State to ACCEPTING_INTERESTS
caller: 01:02:52 : Changing State to WAITING_INTERESTED_COUNT
callee: 01:02:52 : Received: interested_count: amount = "0"
caller: 01:02:52 : interest time ended!
callee: 01:02:52 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:02:52 : Received: interested_count: amount = "0"
caller: 01:02:52 : Item 2 is discarded due to lack of interest
caller: 01:02:52 : Starting Auction for next item
callee: 01:02:52 : interest time ended!
callee: 01:02:52 : Item 2 is discarded due to lack of interest
callee: 01:02:52 : Starting Auction for next item
caller: 01:02:52 : Changing State to ACCEPTING_INTERESTS
callee: 01:02:52 : Changing State to ACCEPTING_INTERESTS
caller: 01:03:02 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:03:02 : interest time ended!
callee: 01:03:02 : Received: interested_count: amount = "0"
callee: 01:03:02 : Changing State to WAITING_INTERESTED_COUNT
callee: 01:03:02 : interest time ended!
caller: 01:03:02 : Received: interested_count: amount = "0"
callee: 01:03:02 : Item 3 is discarded due to lack of interest
caller: 01:03:02 : Item 3 is discarded due to lack of interest
caller: 01:03:02 : Auctions have ended
callee: 01:03:02 : Auctions have ended
caller: 01:03:03 : Shuting Down ...
callee: 01:03:04 : Shuting Down ...
```

Testcase2:

```
5
chara   4444
bill    5556
kostas  4444
maria   5556
giannis 4444

1000  chara   launch
1500  bill    launch
14500 chara   i_am_interested
```

```
14600 bill    i_am_interested
15000 kostas  launch
30000 chara   bid 150
30500 bill    bid 200
30500 maria   launch
31500 giannis launch
32000 kostas  list_high_bid
39000 maria   i_am_interested
40000 kostas  i_am_interested
44000 giannis list_description
50000 maria   bid 560
50010 kostas  bid 600
55000 chara   quit
```

Testcase2 description:

```
Bill: Interested for item 1. Nobody declares interest, so it gets a discount. After
the discount, chara bids 150. Then Bill bids 200 and wins the item. Item 2 is skiped
for Bill, because he doesn't declare interest. The same for item 3.

Maria: i_am_interested before the appropriate time lapse => cannot bid for the
item, cause doesn't exist in the interested bidders list

Chara: bids for item 1 but finally Bill wins it. Then she quits

Kostas: not in the interested users list => cannot bid

Giannis: just asks for descriptions.
```

Testcase2 server log file:

```
caller: 00:59:50 : Querying: Delete from items;
caller: 00:59:50 : Querying: Insert into items values("1", "50", "This is the first item",
 null, null);
caller: 00:59:50 : Querying: Insert into items values("2", "40", "This is the second item",
 null, null);
caller: 00:59:50 : Querying: Insert into items values("3", "100", "This is the last item",
 null, null);
callee: 00:59:50 : Querying: Delete from items;
callee: 00:59:50 : Querying: Insert into items values("1", "50", "This is the first item",
null, null);
callee: 00:59:51 : Querying: Insert into items values("2", "40", "This is the second item",
null, null);
callee: 00:59:51 : Querying: Insert into items values("3", "100", "This is the last item",
null, null);
callee: 00:59:51 : Auctioneer starting ...
caller: 00:59:51 : Auctioneer starting ...
callee: 00:59:51 : Now accepting peers!
caller: 00:59:51 : Connecting to remote peer
```

```
caller: 00:59:52 : Connected to remote peer
callee: 00:59:52 : Peer accepted!
caller: 00:59:52 : Connection accepted!
caller: 00:59:52 : Received: connect: username = "chara@localhost:4444"
callee: 00:59:52 : Connection accepted!
callee: 00:59:52 : Received: connect: username = "bill@localhost:5556"
caller: 00:59:56 : Auctions start in 5 seconds!
callee: 01:00:01 : Auctions are beggining!
caller: 01:00:01 : Auctions are beggining!
callee: 01:00:01 : Changing State to READY_TO_BEGIN
caller: 01:00:01 : Changing State to READY_TO_BEGIN
caller: 01:00:01 : Received: ready_to_run:
callee: 01:00:01 : Received: ready_to_run:
callee: 01:00:01 : Got ready to run message
caller: 01:00:01 : Got ready to run message
caller: 01:00:01 : Starting Auction for next item
callee: 01:00:01 : Starting Auction for next item
callee: 01:00:01 : Changing State to ACCEPTING_INTERESTS
caller: 01:00:01 : Changing State to ACCEPTING_INTERESTS
caller: 01:00:05 : Received: i_am_interested: username = "chara@localhost:4444", item_id = "1"
caller: 01:00:05 : Added user chara@localhost:4444 to interestedUsers list for the current item
callee: 01:00:05 : Received: i_am_interested: username = "bill@localhost:5556", item_id = "1"
callee: 01:00:05 : Added user bill@localhost:5556 to interestedUsers list for the current item
caller: 01:00:06 : Connection accepted!
caller: 01:00:06 : Received: connect: username = "kostas@localhost:4444"
callee: 01:00:11 : Changing State to WAITING_INTERESTED_COUNT
callee: 01:00:11 : interest time ended!
caller: 01:00:11 : Received: interested_count: amount = "1"
caller: 01:00:11 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:00:11 : interest time ended!
callee: 01:00:11 : Received: interested_count: amount = "1"
caller: 01:00:11 : Changing State to ACCEPTING_BIDS
callee: 01:00:11 : Changing State to ACCEPTING_BIDS
caller: 01:00:21 : bid timer ended
caller: 01:00:21 : Changing State to READY_TO_END
callee: 01:00:21 : bid timer ended
callee: 01:00:21 : Changing State to READY_TO_END
caller: 01:00:21 : Received: ready_to_end:
callee: 01:00:21 : Received: ready_to_end:
callee: 01:00:21 : Got ready to end message
caller: 01:00:21 : Got ready to end message
caller: 01:00:21 : sent bid again from handleReadyToEnd
caller: 01:00:21 : Changing State to ACCEPTING_BIDS
callee: 01:00:21 : sent bid again from handleReadyToEnd
callee: 01:00:21 : Changing State to ACCEPTING_BIDS
caller: 01:00:21 : Received: my_bid: amount = "150", username = "chara@localhost:4444",
item_id = "1"
callee: 01:00:21 : Received: got_bid: amount = "150", username = "chara@localhost:4444"
callee: 01:00:21 : New highest bid from user chara@localhost:4444, amount = 150
callee: 01:00:21 : Changing State to ACCEPTING_BIDS
```

```
caller: 01:00:21 : Received: bid_ok: amount = "150", username = "chara@localhost:4444"
caller: 01:00:21 : New highest bid from user chara@localhost:4444, amount = 150
caller: 01:00:21 : Changing State to ACCEPTING_BIDS
callee: 01:00:21 : Received: my_bid: amount = "200", username = "bill@localhost:5556",
item_id = "1"
caller: 01:00:21 : Received: got_bid: amount = "200", username = "bill@localhost:5556"
caller: 01:00:21 : New highest bid from user bill@localhost:5556, amount = 200
caller: 01:00:21 : Changing State to ACCEPTING_BIDS
callee: 01:00:21 : Received: bid_ok: amount = "200", username = "bill@localhost:5556"
callee: 01:00:21 : New highest bid from user bill@localhost:5556, amount = 200
callee: 01:00:21 : Changing State to ACCEPTING_BIDS
callee: 01:00:21 : Connection accepted!
callee: 01:00:21 : Received: connect: username = "maria@localhost:5556"
caller: 01:00:22 : Connection accepted!
caller: 01:00:22 : Received: connect: username = "giannis@localhost:4444"
caller: 01:00:31 : bid timer ended
caller: 01:00:31 : Changing State to READY_TO_END
callee: 01:00:31 : Received: ready_to_end:
callee: 01:00:31 : Got ready to end message
caller: 01:00:31 : Querying: Update items set bid = "200", bidder = "bill@localhost:5556"
where id = "1";
callee: 01:00:31 : bid timer ended
callee: 01:00:31 : Changing State to READY_TO_END
callee: 01:00:31 : Querying: Update items set bid = "200", bidder = "bill@localhost:5556"
where id = "1";
caller: 01:00:31 : Starting Auction for next item
caller: 01:00:31 : Changing State to ACCEPTING_INTERESTS
caller: 01:00:31 : Received: ready_to_end:
caller: 01:00:31 : Got ready to end message
callee: 01:00:31 : Starting Auction for next item
callee: 01:00:31 : Changing State to ACCEPTING_INTERESTS
callee: 01:00:41 : Received: my_bid: amount = "560", username = "maria@localhost:5556",
item_id = "2"
caller: 01:00:41 : Received: my_bid: amount = "600", username = "kostas@localhost:4444",
item_id = "2"
caller: 01:00:41 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:00:41 : interest time ended!
callee: 01:00:41 : Received: interested_count: amount = "0"
callee: 01:00:41 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:00:41 : Received: interested_count: amount = "0"
caller: 01:00:41 : Item 2 is discarded due to lack of interest
caller: 01:00:41 : Starting Auction for next item
callee: 01:00:41 : interest time ended!
callee: 01:00:41 : Item 2 is discarded due to lack of interest
callee: 01:00:41 : Starting Auction for next item
caller: 01:00:41 : Changing State to ACCEPTING_INTERESTS
callee: 01:00:41 : Changing State to ACCEPTING_INTERESTS
caller: 01:00:46 : Received: quit: username = "chara@localhost:4444"
caller: 01:00:46 : User "chara@localhost:4444" quit.
caller: 01:00:51 : Changing State to WAITING_INTERESTED_COUNT
```

```
caller: 01:00:51 : interest time ended!
callee: 01:00:51 : Received: interested_count: amount = "0"
callee: 01:00:51 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:00:51 : Received: interested_count: amount = "0"
caller: 01:00:51 : Item 3 is discarded due to lack of interest
callee: 01:00:51 : interest time ended!
callee: 01:00:51 : Item 3 is discarded due to lack of interest
caller: 01:00:51 : Auctions have ended
callee: 01:00:51 : Auctions have ended
caller: 01:00:52 : Shuting Down ...
callee: 01:00:53 : Shuting Down ...
```

Testcase3:

```
2
chara   4444
bill    5556

1000  chara   launch
1500  bill    launch
10000 chara   i_am_interested
11000 bill    i_am_interested
```

Testcase3 description:

```
Item 1: Chara is registered in the interested users list and gets updates
for the discount. No-one ever tries to purchase item. Bill also gets update
for item 1 (for the discounts as well) but never buys.

Items 2 & 3 get discarded
```

Testcase3 server log file:

```
caller: 00:32:06 : Querying: Delete from items;
caller: 00:32:06 : Querying: Insert into items values("1", "50", "This is the first item",
null, null);
caller: 00:32:06 : Querying: Insert into items values("2", "40", "This is the second item",
null, null);
caller: 00:32:06 : Querying: Insert into items values("3", "100", "This is the last item",
null, null);
callee: 00:32:06 : Querying: Delete from items;
callee: 00:32:06 : Querying: Insert into items values("1", "50", "This is the first item",
null, null);
callee: 00:32:06 : Querying: Insert into items values("2", "40", "This is the second item",
null, null);
callee: 00:32:06 : Querying: Insert into items values("3", "100", "This is the last item",
null, null);
caller: 00:32:06 : Auctioneer starting ...
```

```
callee: 00:32:06 : Auctioneer starting ...
caller: 00:32:06 : Connecting to remote peer
callee: 00:32:06 : Now accepting peers!
caller: 00:32:07 : Connected to remote peer
callee: 00:32:07 : Peer accepted!
caller: 00:32:08 : Connection accepted!
caller: 00:32:08 : Received: connect: username = "chara@localhost:4444"
callee: 00:32:08 : Connection accepted!
callee: 00:32:08 : Received: connect: username = "bill@localhost:5556"
caller: 00:32:11 : Auctions start in 5 seconds!
caller: 00:32:16 : Auctions are beggining!
callee: 00:32:16 : Auctions are beggining!
caller: 00:32:16 : Changing State to READY_TO_BEGIN
callee: 00:32:16 : Changing State to READY_TO_BEGIN
caller: 00:32:16 : Received: ready_to_run:
caller: 00:32:16 : Got ready to run message
caller: 00:32:16 : Starting Auction for next item
callee: 00:32:16 : Received: ready_to_run:
callee: 00:32:16 : Got ready to run message
callee: 00:32:16 : Starting Auction for next item
callee: 00:32:16 : Changing State to ACCEPTING_INTERESTS
caller: 00:32:16 : Changing State to ACCEPTING_INTERESTS
caller: 00:32:17 : Received: i_am_interested: username = "chara@localhost:4444", item_id = "1"
caller: 00:32:17 : Added user chara@localhost:4444 to interestedUsers list for the current item
callee: 00:32:18 : Received: i_am_interested: username = "bill@localhost:5556", item_id = "1"
callee: 00:32:18 : Added user bill@localhost:5556 to interestedUsers list for the current item
callee: 00:32:26 : Changing State to WAITING_INTERESTED_COUNT
callee: 00:32:26 : interest time ended!
caller: 00:32:26 : Received: interested_count: amount = "1"
caller: 00:32:26 : Changing State to WAITING_INTERESTED_COUNT
caller: 00:32:26 : interest time ended!
callee: 00:32:26 : Received: interested_count: amount = "1"
caller: 00:32:26 : Changing State to ACCEPTING_BIDS
callee: 00:32:26 : Changing State to ACCEPTING_BIDS
caller: 00:32:36 : bid timer ended
caller: 00:32:36 : Changing State to READY_TO_END
callee: 00:32:36 : Received: ready_to_end:
callee: 00:32:36 : Got ready to end message
callee: 00:32:36 : bid timer ended
callee: 00:32:36 : Changing State to READY_TO_END
caller: 00:32:36 : Received: ready_to_end:
caller: 00:32:36 : Got ready to end message
caller: 00:32:36 : sent bid again from handleReadyToEnd
caller: 00:32:36 : Changing State to ACCEPTING_BIDS
callee: 00:32:36 : sent bid again message
callee: 00:32:36 : Changing State to ACCEPTING_BIDS
caller: 00:32:46 : bid timer ended
caller: 00:32:46 : Changing State to READY_TO_END
callee: 00:32:46 : Received: ready_to_end:
callee: 00:32:46 : Got ready to end message
```

```
caller: 00:32:46 : sent bid again message
caller: 00:32:46 : Changing State to ACCEPTING_BIDS
callee: 00:32:46 : bid timer ended
callee: 00:32:46 : Changing State to READY_TO_END
caller: 00:32:46 : Received: ready_to_end:
caller: 00:32:46 : Got ready to end message
callee: 00:32:46 : sent bid again message
callee: 00:32:46 : Changing State to ACCEPTING_BIDS
caller: 00:32:56 : bid timer ended
caller: 00:32:56 : Changing State to READY_TO_END
callee: 00:32:56 : Received: ready_to_end:
callee: 00:32:56 : Got ready to end message
caller: 00:32:56 : sent bid again message
caller: 00:32:56 : Changing State to ACCEPTING_BIDS
callee: 00:32:56 : bid timer ended
callee: 00:32:56 : Changing State to READY_TO_END
caller: 00:32:56 : Received: ready_to_end:
caller: 00:32:56 : Got ready to end message
callee: 00:32:56 : sent bid again message
callee: 00:32:56 : Changing State to ACCEPTING_BIDS
caller: 00:33:06 : bid timer ended
caller: 00:33:06 : Changing State to READY_TO_END
caller: 00:33:06 : sent bid again message
caller: 00:33:06 : Changing State to ACCEPTING_BIDS
callee: 00:33:06 : Received: ready_to_end:
callee: 00:33:06 : Got ready to end message
callee: 00:33:06 : bid timer ended
callee: 00:33:06 : Changing State to READY_TO_END
caller: 00:33:06 : Received: ready_to_end:
caller: 00:33:06 : Got ready to end message
callee: 00:33:06 : sent bid again message
callee: 00:33:06 : Changing State to ACCEPTING_BIDS
caller: 00:33:16 : bid timer ended
caller: 00:33:16 : Changing State to READY_TO_END
callee: 00:33:16 : Received: ready_to_end:
callee: 00:33:16 : Got ready to end message
caller: 00:33:16 : Querying: Update items set bid = "30", bidder = "no_holder" where id = "1";
callee: 00:33:16 : bid timer ended
callee: 00:33:16 : Changing State to READY_TO_END
callee: 00:33:16 : Querying: Update items set bid = "30", bidder = "no_holder" where id = "1";
caller: 00:33:17 : Starting Auction for next item
caller: 00:33:17 : Changing State to ACCEPTING_INTERESTS
caller: 00:33:17 : Received: ready_to_end:
caller: 00:33:17 : Got ready to end message
callee: 00:33:17 : Starting Auction for next item
callee: 00:33:17 : Changing State to ACCEPTING_INTERESTS
caller: 00:33:27 : Changing State to WAITING_INTERESTED_COUNT
callee: 00:33:27 : Received: interested_count: amount = "0"
caller: 00:33:27 : interest time ended!
callee: 00:33:27 : Changing State to WAITING_INTERESTED_COUNT
```

```
caller: 00:33:27 : Received: interested_count: amount = "0"
caller: 00:33:27 : Item 2 is discarded due to lack of interest
caller: 00:33:27 : Starting Auction for next item
caller: 00:33:27 : Changing State to ACCEPTING_INTERESTS
callee: 00:33:27 : interest time ended!
callee: 00:33:27 : Item 2 is discarded due to lack of interest
callee: 00:33:27 : Starting Auction for next item
callee: 00:33:27 : Changing State to ACCEPTING_INTERESTS
caller: 00:33:37 : Changing State to WAITING_INTERESTED_COUNT
caller: 00:33:37 : interest time ended!
callee: 00:33:37 : Received: interested_count: amount = "0"
callee: 00:33:37 : Changing State to WAITING_INTERESTED_COUNT
callee: 00:33:37 : interest time ended!
callee: 00:33:37 : Item 3 is discarded due to lack of interest
caller: 00:33:37 : Received: interested_count: amount = "0"
caller: 00:33:37 : Item 3 is discarded due to lack of interest
caller: 00:33:37 : Auctions have ended
callee: 00:33:37 : Auctions have ended
callee: 00:33:38 : Shuting Down ...
caller: 00:33:39 : Shuting Down ...
```

Testcase4:

```
5
chara   5556
bill    5556
kostas  4444
maria   5556
giannis 4444

1000  chara   launch
1500  bill    launch
1500  kostas  launch
1500  giannis launch
1500  maria   launch
14500 chara   i_am_interested
14500 bill    i_am_interested
14500 kostas  i_am_interested
14500 maria   i_am_interested
14500 giannis i_am_interested
25000 bill    bid 600
25000 maria   bid 560
25000 chara   bid 700
25000 kostas  bid 710
25000 giannis bid 600
27000 maria   list_high_bid
```

Testcase4 description:

```
5 users connect at the same time, declare interest at the same time and bid different amounts
at the same time at different auctioneers.
```

Testcase4 server log file:

```
caller: 01:04:36 : Querying: Delete from items;
caller: 01:04:36 : Querying: Insert into items values("1", "50", "This is the first item",
null, null);
caller: 01:04:36 : Querying: Insert into items values("2", "40", "This is the second item",
null, null);
caller: 01:04:36 : Querying: Insert into items values("3", "100", "This is the last item",
null, null);
callee: 01:04:36 : Querying: Delete from items;
callee: 01:04:36 : Querying: Insert into items values("1", "50", "This is the first item",
null, null);
callee: 01:04:36 : Querying: Insert into items values("2", "40", "This is the second item",
null, null);
callee: 01:04:36 : Querying: Insert into items values("3", "100", "This is the last item",
null, null);
caller: 01:04:36 : Auctioneer starting ...
caller: 01:04:36 : Connecting to remote peer
callee: 01:04:36 : Auctioneer starting ...
callee: 01:04:36 : Now accepting peers!
caller: 01:04:37 : Connected to remote peer
callee: 01:04:37 : Peer accepted!
callee: 01:04:38 : Connection accepted!
callee: 01:04:38 : Received: connect: username = "chara@localhost:5556"
callee: 01:04:38 : Connection accepted!
callee: 01:04:38 : Received: connect: username = "bill@localhost:5556"
callee: 01:04:38 : Connection accepted!
caller: 01:04:38 : Connection accepted!
caller: 01:04:38 : Received: connect: username = "kostas@localhost:4444"
caller: 01:04:38 : Connection accepted!
caller: 01:04:38 : Received: connect: username = "giannis@localhost:4444"
callee: 01:04:38 : Received: connect: username = "maria@localhost:5556"
caller: 01:04:41 : Auctions start in 5 seconds!
caller: 01:04:46 : Auctions are beggining!
caller: 01:04:46 : Changing State to READY_TO_BEGIN
callee: 01:04:46 : Received: ready_to_run:
callee: 01:04:46 : Got ready to run message
callee: 01:04:46 : Auctions are beggining!
callee: 01:04:46 : Changing State to READY_TO_BEGIN
callee: 01:04:46 : Starting Auction for next item
caller: 01:04:46 : Received: ready_to_run:
caller: 01:04:46 : Got ready to run message
caller: 01:04:46 : Starting Auction for next item
caller: 01:04:46 : Changing State to ACCEPTING_INTERESTS
```

```
callee: 01:04:46 : Changing State to ACCEPTING_INTERESTS
callee: 01:04:51 : Received: i_am_interested: username = "bill@localhost:5556",
item_id = "1"
callee: 01:04:51 : Added user bill@localhost:5556 to interestedUsers list for the
current item
caller: 01:04:51 : Received: i_am_interested: username = "giannis@localhost:4444",
item_id = "1"
caller: 01:04:51 : Added user giannis@localhost:4444 to interestedUsers list for the
current item
caller: 01:04:51 : Received: i_am_interested: username = "kostas@localhost:4444",
item_id = "1"
caller: 01:04:51 : Added user kostas@localhost:4444 to interestedUsers list for the
current item
callee: 01:04:51 : Received: i_am_interested: username = "maria@localhost:5556",
item_id = "1"
callee: 01:04:51 : Added user maria@localhost:5556 to interestedUsers list for the
current item
callee: 01:04:51 : Received: i_am_interested: username = "chara@localhost:5556",
item_id = "1"
callee: 01:04:51 : Added user chara@localhost:5556 to interestedUsers list for the
current item
caller: 01:04:56 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:04:56 : interest time ended!
callee: 01:04:56 : Received: interested_count: amount = "2"
callee: 01:04:56 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:04:56 : Received: interested_count: amount = "3"
caller: 01:04:56 : Changing State to ACCEPTING_BIDS
callee: 01:04:56 : interest time ended!
callee: 01:04:56 : Changing State to ACCEPTING_BIDS
callee: 01:05:02 : Received: my_bid: amount = "560", username = "maria@localhost:5556",
item_id = "1"
caller: 01:05:02 : Received: my_bid: amount = "710", username = "kostas@localhost:4444",
item_id = "1"
callee: 01:05:02 : Received: my_bid: amount = "700", username = "chara@localhost:5556",
item_id = "1"
caller: 01:05:02 : Received: got_bid: amount = "560", username = "maria@localhost:5556"
caller: 01:05:02 : Received: my_bid: amount = "600", username = "giannis@localhost:4444",
item_id = "1"
callee: 01:05:02 : Received: my_bid: amount = "600", username = "bill@localhost:5556",
item_id = "1"
caller: 01:05:02 : Received: got_bid: amount = "700", username = "chara@localhost:5556"
callee: 01:05:02 : Received: got_bid: amount = "710", username = "kostas@localhost:4444"
callee: 01:05:02 : New highest bid from user kostas@localhost:4444, amount = 710
callee: 01:05:02 : Changing State to ACCEPTING_BIDS
caller: 01:05:02 : Received: bid_ok: amount = "710", username = "kostas@localhost:4444"
caller: 01:05:02 : New highest bid from user kostas@localhost:4444, amount = 710
caller: 01:05:02 : Changing State to ACCEPTING_BIDS
callee: 01:05:12 : bid timer ended
callee: 01:05:12 : Changing State to READY_TO_END
caller: 01:05:12 : Received: ready_to_end:
```

```
caller: 01:05:12 : Got ready to end message
caller: 01:05:12 : bid timer ended
caller: 01:05:12 : Changing State to READY_TO_END
callee: 01:05:12 : Received: ready_to_end:
callee: 01:05:12 : Got ready to end message
callee: 01:05:12 : Querying: Update items set bid = "710", bidder = "kostas@localhost:4444"
where id = "1";
caller: 01:05:12 : Querying: Update items set bid = "710", bidder = "kostas@localhost:4444"
 where id = "1";
callee: 01:05:12 : Starting Auction for next item
callee: 01:05:12 : Changing State to ACCEPTING_INTERESTS
caller: 01:05:12 : Starting Auction for next item
caller: 01:05:12 : Changing State to ACCEPTING_INTERESTS
callee: 01:05:22 : Changing State to WAITING_INTERESTED_COUNT
callee: 01:05:22 : interest time ended!
caller: 01:05:22 : Received: interested_count: amount = "0"
caller: 01:05:22 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:05:22 : interest time ended!
callee: 01:05:22 : Received: interested_count: amount = "0"
caller: 01:05:22 : Item 2 is discarded due to lack of interest
caller: 01:05:22 : Starting Auction for next item
callee: 01:05:22 : Item 2 is discarded due to lack of interest
callee: 01:05:22 : Starting Auction for next item
callee: 01:05:22 : Changing State to ACCEPTING_INTERESTS
caller: 01:05:22 : Changing State to ACCEPTING_INTERESTS
caller: 01:05:32 : Changing State to WAITING_INTERESTED_COUNT
callee: 01:05:32 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:05:32 : interest time ended!
callee: 01:05:32 : interest time ended!
caller: 01:05:32 : Received: interested_count: amount = "0"
caller: 01:05:32 : Item 3 is discarded due to lack of interest
caller: 01:05:32 : Auctions have ended
callee: 01:05:32 : Received: interested_count: amount = "0"
callee: 01:05:32 : Item 3 is discarded due to lack of interest
callee: 01:05:32 : Auctions have ended
caller: 01:05:33 : Shuting Down ...
callee: 01:05:34 : Shuting Down ...
```

Testcase5:

```
2
chara   4444
bill    5556

1000  chara   launch
1500  bill    launch
10000 chara   i_am_interested
11000 bill    i_am_interested
40000 chara   bid 41
40500 bill    bid 42
```

```
40600 chara   bid 43
40700 bill    bid 44
41000 chara   bid 45
41100 chara   bid 46
42000 bill    bid 47
42300 chara   bid 48
43000 bill    bid 49
```

Testcase5 description:

```
Item 1: In the beginning, both chara & bill are interested. After that, nobody bids,
so we have the first discount of the item. Sequence of chara's and bill's bids,
one higher than the other. Final winner announced is bill.
```

Testcase5 server log file:

```
caller: 01:40:15 : Querying: Delete from items;
caller: 01:40:15 : Querying: Insert into items values("1", "50", "This is the first item",
null, null);
caller: 01:40:15 : Querying: Insert into items values("2", "40", "This is the second item",
null, null);
caller: 01:40:15 : Querying: Insert into items values("3", "100", "This is the last item",
null, null);
callee: 01:40:16 : Querying: Delete from items;
callee: 01:40:16 : Querying: Insert into items values("1", "50", "This is the first item",
null, null);
callee: 01:40:16 : Querying: Insert into items values("2", "40", "This is the second item",
null, null);
callee: 01:40:16 : Querying: Insert into items values("3", "100", "This is the last item",
null, null);
caller: 01:40:16 : Auctioneer starting ...
callee: 01:40:16 : Auctioneer starting ...
caller: 01:40:16 : Connecting to remote peer
callee: 01:40:16 : Now accepting peers!
caller: 01:40:17 : Connected to remote peer
callee: 01:40:17 : Peer accepted!
caller: 01:40:17 : Connection accepted!
caller: 01:40:17 : Received: connect: username = "chara@localhost:4444"
callee: 01:40:18 : Connection accepted!
callee: 01:40:18 : Received: connect: username = "bill@localhost:5556"
caller: 01:40:21 : Auctions start in 5 seconds!
caller: 01:40:26 : Auctions are beggining!
callee: 01:40:26 : Auctions are beggining!
callee: 01:40:26 : Changing State to READY_TO_BEGIN
caller: 01:40:26 : Changing State to READY_TO_BEGIN
callee: 01:40:26 : Received: ready_to_run:
caller: 01:40:26 : Received: ready_to_run:
caller: 01:40:26 : Got ready to run message
callee: 01:40:26 : Got ready to run message
```

```
callee: 01:40:26 : Starting Auction for next item
caller: 01:40:26 : Starting Auction for next item
caller: 01:40:26 : Changing State to ACCEPTING_INTERESTS
callee: 01:40:26 : Changing State to ACCEPTING_INTERESTS
caller: 01:40:26 : Received: i_am_interested: username = "chara@localhost:4444", item_id = "1"
caller: 01:40:26 : Added user chara@localhost:4444 to interestedUsers list for the current item
callee: 01:40:27 : Received: i_am_interested: username = "bill@localhost:5556", item_id = "1"
callee: 01:40:27 : Added user bill@localhost:5556 to interestedUsers list for the current item
caller: 01:40:36 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:40:36 : interest time ended!
callee: 01:40:36 : Received: interested_count: amount = "1"
callee: 01:40:36 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:40:36 : Received: interested_count: amount = "1"
caller: 01:40:36 : Changing State to ACCEPTING_BIDS
callee: 01:40:36 : interest time ended!
callee: 01:40:36 : Changing State to ACCEPTING_BIDS
caller: 01:40:46 : bid timer ended
caller: 01:40:46 : Changing State to READY_TO_END
callee: 01:40:46 : Received: ready_to_end:
callee: 01:40:46 : Got ready to end message
callee: 01:40:46 : bid timer ended
callee: 01:40:46 : Changing State to READY_TO_END
caller: 01:40:46 : Received: ready_to_end:
caller: 01:40:46 : Got ready to end message
caller: 01:40:46 : sent bid again from handleReadyToEnd
caller: 01:40:46 : Changing State to ACCEPTING_BIDS
callee: 01:40:46 : sent bid again message
callee: 01:40:46 : Changing State to ACCEPTING_BIDS
caller: 01:40:56 : bid timer ended
caller: 01:40:56 : Changing State to READY_TO_END
callee: 01:40:56 : Received: ready_to_end:
callee: 01:40:56 : Got ready to end message
caller: 01:40:56 : sent bid again message
callee: 01:40:56 : bid timer ended
callee: 01:40:56 : Changing State to READY_TO_END
caller: 01:40:56 : Changing State to ACCEPTING_BIDS
caller: 01:40:56 : Received: ready_to_end:
caller: 01:40:56 : Got ready to end message
callee: 01:40:56 : sent bid again message
callee: 01:40:56 : Changing State to ACCEPTING_BIDS
caller: 01:40:56 : Received: my_bid: amount = "41", username = "chara@localhost:4444",
item_id = "1"
callee: 01:40:56 : Received: got_bid: amount = "41", username = "chara@localhost:4444"
callee: 01:40:56 : New highest bid from user chara@localhost:4444, amount = 41
callee: 01:40:56 : Changing State to ACCEPTING_BIDS
caller: 01:40:56 : Received: bid_ok: amount = "41", username = "chara@localhost:4444"
caller: 01:40:56 : New highest bid from user chara@localhost:4444, amount = 41
caller: 01:40:56 : Changing State to ACCEPTING_BIDS
callee: 01:40:56 : Received: my_bid: amount = "42", username = "bill@localhost:5556",
item_id = "1"
```

```
caller: 01:40:57 : Received: got_bid: amount = "42", username = "bill@localhost:5556"
caller: 01:40:57 : New highest bid from user bill@localhost:5556, amount = 42
caller: 01:40:57 : Changing State to ACCEPTING_BIDS
callee: 01:40:57 : Received: bid_ok: amount = "42", username = "bill@localhost:5556"
callee: 01:40:57 : New highest bid from user bill@localhost:5556, amount = 42
callee: 01:40:57 : Changing State to ACCEPTING_BIDS
caller: 01:40:57 : Received: my_bid: amount = "43", username = "chara@localhost:4444",
item_id = "1"
callee: 01:40:57 : Received: got_bid: amount = "43", username = "chara@localhost:4444"
callee: 01:40:57 : New highest bid from user chara@localhost:4444, amount = 43
callee: 01:40:57 : Changing State to ACCEPTING_BIDS
caller: 01:40:57 : Received: bid_ok: amount = "43", username = "chara@localhost:4444"
caller: 01:40:57 : New highest bid from user chara@localhost:4444, amount = 43
caller: 01:40:57 : Changing State to ACCEPTING_BIDS
callee: 01:40:57 : Received: my_bid: amount = "44", username = "bill@localhost:5556",
item_id = "1"
caller: 01:40:57 : Received: got_bid: amount = "44", username = "bill@localhost:5556"
caller: 01:40:57 : New highest bid from user bill@localhost:5556, amount = 44
caller: 01:40:57 : Changing State to ACCEPTING_BIDS
callee: 01:40:57 : Received: bid_ok: amount = "44", username = "bill@localhost:5556"
callee: 01:40:57 : New highest bid from user bill@localhost:5556, amount = 44
callee: 01:40:57 : Changing State to ACCEPTING_BIDS
caller: 01:40:57 : Received: my_bid: amount = "45", username = "chara@localhost:4444",
item_id = "1"
callee: 01:40:57 : Received: got_bid: amount = "45", username = "chara@localhost:4444"
callee: 01:40:57 : New highest bid from user chara@localhost:4444, amount = 45
callee: 01:40:57 : Changing State to ACCEPTING_BIDS
caller: 01:40:57 : Received: bid_ok: amount = "45", username = "chara@localhost:4444"
caller: 01:40:57 : New highest bid from user chara@localhost:4444, amount = 45
caller: 01:40:57 : Changing State to ACCEPTING_BIDS
caller: 01:40:57 : Received: my_bid: amount = "46", username = "chara@localhost:4444",
item_id = "1"
callee: 01:40:57 : Received: got_bid: amount = "46", username = "chara@localhost:4444"
callee: 01:40:57 : New highest bid from user chara@localhost:4444, amount = 46
callee: 01:40:57 : Changing State to ACCEPTING_BIDS
caller: 01:40:57 : Received: bid_ok: amount = "46", username = "chara@localhost:4444"
caller: 01:40:57 : New highest bid from user chara@localhost:4444, amount = 46
caller: 01:40:57 : Changing State to ACCEPTING_BIDS
callee: 01:40:58 : Received: my_bid: amount = "47", username = "bill@localhost:5556",
item_id = "1"
caller: 01:40:58 : Received: got_bid: amount = "47", username = "bill@localhost:5556"
caller: 01:40:58 : New highest bid from user bill@localhost:5556, amount = 47
caller: 01:40:58 : Changing State to ACCEPTING_BIDS
callee: 01:40:58 : Received: bid_ok: amount = "47", username = "bill@localhost:5556"
callee: 01:40:58 : New highest bid from user bill@localhost:5556, amount = 47
callee: 01:40:58 : Changing State to ACCEPTING_BIDS
caller: 01:40:58 : Received: my_bid: amount = "48", username = "chara@localhost:4444",
item_id = "1"
callee: 01:40:58 : Received: got_bid: amount = "48", username = "chara@localhost:4444"
callee: 01:40:58 : New highest bid from user chara@localhost:4444, amount = 48
```

```
callee: 01:40:58 : Changing State to ACCEPTING_BIDS
caller: 01:40:58 : Received: bid_ok: amount = "48", username = "chara@localhost:4444"
caller: 01:40:58 : New highest bid from user chara@localhost:4444, amount = 48
caller: 01:40:58 : Changing State to ACCEPTING_BIDS
callee: 01:40:59 : Received: my_bid: amount = "49", username = "bill@localhost:5556",
item_id = "1"
caller: 01:40:59 : Received: got_bid: amount = "49", username = "bill@localhost:5556"
caller: 01:40:59 : New highest bid from user bill@localhost:5556, amount = 49
caller: 01:40:59 : Changing State to ACCEPTING_BIDS
callee: 01:40:59 : Received: bid_ok: amount = "49", username = "bill@localhost:5556"
callee: 01:40:59 : New highest bid from user bill@localhost:5556, amount = 49
callee: 01:40:59 : Changing State to ACCEPTING_BIDS
caller: 01:41:09 : bid timer ended
caller: 01:41:09 : Changing State to READY_TO_END
callee: 01:41:09 : Received: ready_to_end:
callee: 01:41:09 : Got ready to end message
callee: 01:41:09 : bid timer ended
callee: 01:41:09 : Changing State to READY_TO_END
caller: 01:41:09 : Querying: Update items set bid = "49", bidder = "bill@localhost:5556"
where id = "1";
callee: 01:41:09 : Querying: Update items set bid = "49", bidder = "bill@localhost:5556"
where id = "1";
caller: 01:41:09 : Starting Auction for next item
caller: 01:41:09 : Changing State to ACCEPTING_INTERESTS
caller: 01:41:09 : Received: ready_to_end:
caller: 01:41:09 : Got ready to end message
callee: 01:41:09 : Starting Auction for next item
callee: 01:41:09 : Changing State to ACCEPTING_INTERESTS
caller: 01:41:19 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:41:19 : interest time ended!
callee: 01:41:19 : Received: interested_count: amount = "0"
callee: 01:41:19 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:41:19 : Received: interested_count: amount = "0"
caller: 01:41:19 : Item 2 is discarded due to lack of interest
caller: 01:41:19 : Starting Auction for next item
caller: 01:41:19 : Changing State to ACCEPTING_INTERESTS
callee: 01:41:19 : interest time ended!
callee: 01:41:19 : Item 2 is discarded due to lack of interest
callee: 01:41:19 : Starting Auction for next item
callee: 01:41:19 : Changing State to ACCEPTING_INTERESTS
caller: 01:41:29 : Changing State to WAITING_INTERESTED_COUNT
caller: 01:41:29 : interest time ended!
callee: 01:41:29 : Received: interested_count: amount = "0"
callee: 01:41:29 : Changing State to WAITING_INTERESTED_COUNT
callee: 01:41:29 : interest time ended!
caller: 01:41:29 : Received: interested_count: amount = "0"
callee: 01:41:29 : Item 3 is discarded due to lack of interest
caller: 01:41:29 : Item 3 is discarded due to lack of interest
caller: 01:41:29 : Auctions have ended
callee: 01:41:29 : Auctions have ended
```

```
callee: 01:41:30 : Shuting Down ...
caller: 01:41:30 : Shuting Down ...
```