Εργασία Δικτυακού Προγραμματισμού

Request-Reply Messaging App

Στόχος της εργασίας είναι η εξοικείωση με το δικτυακό προγραμματισμό χρησιμοποιώντας Sockets, Input/Output streams και Threads (ή RMI) με σκοπό την υλοποίηση προγραμμάτων που βασίζονται στο μοντέλο πελάτη-εξυπηρετητή. Στα πλαίσια αυτής της εργασίας θα αναπτυχθεί ένα κατανεμημένο σύστημα ανταλλαγής μηνυμάτων που χρησιμοποιεί ένα απλό request-reply protocol. Οι clients στέλνουν στον Server ένα Request και έπειτα ο Server απαντάει με ένα Response και η σύνδεση τους τερματίζει.

Στο σύστημα διαφορετικοί χρήστες θα μπορούν να δημιουργήσουν accounts και να στείλουν μηνύματα μεταξύ τους. Τη λειτουργικότητα αυτή θα τους την παρέχουν:

α) ένα πρόγραμμα εξυπηρετητή, το οποίο θα έχει την ικανότητα να διαχειριστεί ταυτόχρονα πολλαπλές αιτήσεις από πελάτες.

β) προγράμματα πελάτη, κάθε ένα από τα οποία θα έχει την ικανότητα να στέλνει αιτήσεις στον εξυπηρετητή.

Τα κύρια στοιχεία του προγράμματος παρουσιάζονται παρακάτω.

Message

Κάθε message που αποστέλλεται, παραλαμβάνεται ή είναι αποθηκευμένο στον εξυπηρετητή πρέπει να έχει τα εξής πεδία:

Ιδιότητα	**Περιγραφή**
boolean isRead	Υποδεικνύει αν το μήνυμα έχει ήδη διαβαστεί.
String sender	Ο αποστολέας του μηνύματος.
String receiver	Ο παραλήπτης του μηνύματος.
String body	Το κείμενο του μηνύματος.

Μπορείτε να συμπληρώσετε οποιαδήποτε άλλη ιδιότητα θεωρείτε απαραίτητη για την υλοποίηση του συστήματος.

Account

Κάθε λογαριασμός χρήστη θα αποθηκεύεται στον εξυπηρετητή έχοντας την εξής μορφή:

Ιδιότητα	**Περιγραφή**
String username	Το όνομα χρήστη. Αποτελείται μόνο από αλφαριθμητικά και τον ειδικό χαρακτήρα “_”.
Int authToken	Ενας μοναδικός αριθμός αναγνώρισης του χρήστη (δημιουργείται από τον server και είναι προσωπικός/κρυφός).
List<Message> messageBox	Το γραμματοκιβώτιο του χρήστη, το οποίο είναι μία λίστα από Messages.

Μπορείτε να συμπληρώσετε οποιαδήποτε άλλη ιδιότητα θεωρείτε απαραίτητη για την υλοποίηση του συστήματος.

MessagingClient

Στο πρόγραμμα-πελάτη υλοποιείται η επικοινωνία του χρήστη με τον εξυπηρετητή.

Το πρόγραμμα θα δέχεται input από τον χρήστη αποστέλλοντας το στον εξυπηρετητή, ενώ ταυτόχρονα θα λαμβάνει δεδομένα από τον εξυπηρετητή και θα τα προβάλλει κατάλληλα στον χρήστη.

MessagingServer

Ο εξυπηρετητής θα τρέχει συνεχώς ως υπηρεσία “ακούγοντας” για εισερχόμενες αιτήσεις από πελάτες. Κάθε εισερχόμενη αίτηση θα ανατίθεται σε διαφορετικό νήμα (μπορείτε να χρησιμοποιήσετε Threads ή τον μηχανισμό του RMI), προκειμένου να υπάρχει η δυνατότητα να εξυπηρετηθούν πολλαπλές αιτήσεις ταυτόχρονα.
Επίσης, στον εξυπηρετητή θα υπάρχει μία λίστα από λογαριασμούς (Account), έτσι ώστε να διατηρούνται δεδομένα όπως οι καταχωρημένοι χρήστες, οι κωδικοί τους και τα γραμματοκιβώτια τους. Τέλος, πρέπει να υλοποιηθούν μέθοδοι για την επικοινωνία με τον χρήστη.

Λειτουργίες

Τα παραδοτέα προγράμματα πρέπει να εκτελούν οπωσδήποτε τις λειτουργίες που θα περιγράψουμε στη συνέχεια. Τα εκτελέσιμα προγράμματα πρέπει να δίνουν

αποτελέσματα **ακριβώς στην μορφή την οποία περιγράφουμε στους πίνακες με τα Σενάρια.**

Server

Ο server θα πρέπει να εκτελείται με τον εξής τρόπο:

```
java server <port number>
```

όπου Port Number η πόρτα στην οποία θα ακούει αιτήσεις.

Client

Ο Client είναι η διεπαφή του χρήστη με το πρόγραμμα. Η γενική εντολή εκτέλεσης είναι:

```
java client <ip> <port number> <FN_ID> <args>
```

όπου

- ip: Η διεύθυνση IP του Server
- port number: Η port στην οποία ακούει ο Server
- FN_ID: Το αναγνωριστικό της λειτουργίας που θα εκτελεστεί
- args: οι παράμετροι της λειτουργίας

Παρέχονται οι παρακάτω λειτουργίες:

Create Account (FN_ID: 1)

`java client <ip> <port number> 1 <username>`

Δημιουργεί ένα account για το user και χρησιμοποιεί το δοσμένο username.
Η συνάρτηση επιστρέφει ένα μοναδικό κωδικό (token) ο οποίος χρησιμοποιείται για να αυθεντικοποιηθεί ο χρήστης στα επόμενα αιτήματα του.

Σενάριο	**Εκτύπωση**
Επιτυχία	<integer>

Παράδειγμα:
`$>java client localhost 5000 1 tester`
`1024`
Στο παραπάνω παράδειγμα ο χρήστης tester, από εδώ και πέρα, θα χρησιμοποιήσει ως auth Token τον αριθμό `1024` στα επόμενα αιτήματα του.	

Ο χρήστης ήδη υπάρχει	Sorry, the user already exists

Παράδειγμα:
`$>java client localhost 5000 1 tester`
`Sorry, the user already exists`	

Λαθος μορφη username	Invalid Username
`Παράδειγμα`	

```
$>java client localhost 5000 1 inv@-lid
Invalid Username
```

Show Accounts (FN_ID: 2)

java client <ip> <port number> 2 <authToken>

Δείχνει μια λίστα με όλα τα accounts που υπάρχουν στο σύστημα.

Σενάριο	**Εκτύπωση**
Σε όλες τις περιπτώσεις εκτυπώνει τη λίστα χρηστών με τη μορφή που δίνεται	1. <username_1> 2. <username_2> … n. <username_n>

Παράδειγμα:
`$>java client localhost 5000 2 1024`
`1. demo`
`2. raven_13`
`3. dr4g0n_sl4y3r`	

Send Message (FN_ID: 3)

java client <ip> <port number> 3 <authToken> <recipient> <message_body>

Στέλνει μήνυμα (<message_body>) στο account με username <recipient>.

Σενάριο	**Εκτύπωση**
Επιτυχής αποστολή	OK

Παράδειγμα:
`$>java client localhost 5000 3 1024 tester "HELLO WORLD"`
`OK`	

Εάν το προφίλ του χρήστη <recipient> δεν υπάρχει	User does not exist

Παράδειγμα:
`$>java client localhost 5000 3 1024 friend "HELLO WORLD"`
`User does not exist`	

Show Inbox (FN_ID: 4)

`java client <ip> <port number> 4 <authToken>`

Εμφανίζει τη λίστα με όλα τα μηνύματα για έναν συγκεκριμένο χρήστη.

Δείχνει μια λίστα με όλα τα μηνύματα που υπάρχουν στο messagebox του χρήστη. Η μορφή που πρέπει να τα εκτυπώνει είναι η εξής:

Σενάριο	**Εκτύπωση**
Επιτυχία	<message_id_1>. from:<username_x> *? <message_id_2>. from:<username_y> *? … <message_id_n>. from:<username_z> *?

Ο αστερίσκος θα είναι ορατός μόνο αν το μήνυμα αυτό δεν έχει διαβαστεί.
Παράδειγμα:
`$>java client localhost 5000 4 1024`
`27. from: raven_13*`
`43. from: demo*`
`55. from: raven_13*`
`58. from: raven_13*`
`67. from: dr4g0n_sl4y3r`
 	
Στο παράδειγμα μας το μήνυμα του χρήστη `dr4g0n_sl4y3r` έχει ήδη διαβαστεί.	

ReadMessage (FN_ID: 5)

`java client <ip> <port number> 5 <authToken> <message_id>`

Αυτή η λειτουργία εμφανίζει το περιεχόμενο ενός μηνύματος του χρήστη με id <message_id>. Έπειτα το μήνυμα μαρκάρεται ως διαβασμένο.

Αν υπάρχει το μήνυμα το πρόγραμμα εκτυπώνει τα παρακάτω

Σενάριο	**Εκτύπωση**
Επιτυχία	(<sender>) <message>

Παράδειγμα:
`$>java client localhost 5000 5 1024 43`
`(demo)Hello World`	

Το μήνυμα δεν υπάρχει	Message ID does not exist

Παράδειγμα:
`$>java client localhost 5000 5 1024 1111`
`Message ID does not exist`	

DeleteMessage (FN_ID: 6)

`java client <ip> <port number> 6 <authToken> <message_id>`

Αυτή η λειτουργία διαγράφει το μήνυμα με id <message_id>.

Σενάριο	**Εκτύπωση**
Επιτυχία	OK

Παράδειγμα:
`$>java client localhost 5000 5 1024 43`
`OK`	

Το μήνυμα δεν υπάρχει	Message does not exist

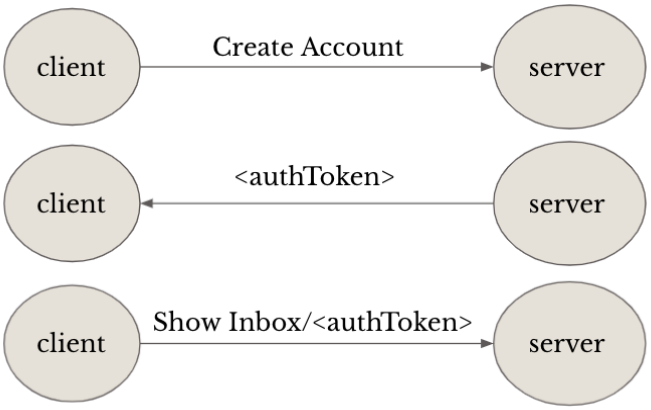
Παράδειγμα:
`$>java client localhost 5000 5 1024 1111`
`Message does not exist`	

Auth Token

Γενικότερα, εφόσον το πρωτόκολλο επικοινωνίας που θα δημιουργηθεί δεν διατηρεί στοιχεία για το χρήστη σε κάθε session θα πρέπει να εξασφαλίσουμε ότι:

- κανένας χρήστης δεν μπορεί να διαβάσει δεδομένα άλλου χρήστη.
- κανένας χρήστης δεν μπορεί να στείλει μηνύματα ως άλλος χρήστης.

Για το λόγο αυτό χρησιμοποιούμε το Auth Token σε κάθε Request. Ο Server πρεπει να συσχετίσει κάθε auth token με ένα Account (κατά τη δημιουργία του account).



Σε περίπτωση που δοθεί λάθος Auth Token (δηλαδή το authToken δεν αντιστοιχίζεται σε κάποιο χρήστη) όλες οι συναρτήσεις πρέπει να επιστρέφουν το αντίστοιχο μήνυμα.

Σενάριο	**Εκτύπωση**
Λάθος authToken	*Invalid Auth Token*

Παραδοτέα

Να παραδοθεί ένα αρχείο zip με λατινικούς χαρακτήρες και φορματ:

<aem>_<name>_<surname>.zip

π.χ. 1243_petros_riginos.zip

το οποίο να περιέχει τα παρακάτω:

- src/: Αρχεία πηγαίου κώδικα.
- jars/:
  - Client.jar: Αρχείο εκτέλεσης του Client
  - Server.jar: Αρχείο εκτέλεσης του Server
- README.md Ένα αρχείο κειμένου, στο οποίο θα περιγράφετε σύντομα τις κλάσεις που υλοποιήσατε και τις υποθέσεις που κάνατε για την υλοποίηση του συστήματος.

Προσοχή: Είναι σημαντικό τα παραπάνω να **υποβληθούν αυστηρά στο σωστό φορματ και δομή φακέλων** προκειμένου να ολοκληρωθεί η διαδικασία υποβολής ομαλά. Σας προτείνουμε να εκτελέσετε τα προγράμματα σας ως jar αρχεία για να επιβεβαιώσετε την ορθή λειτουργία τους πριν τα στείλετε.

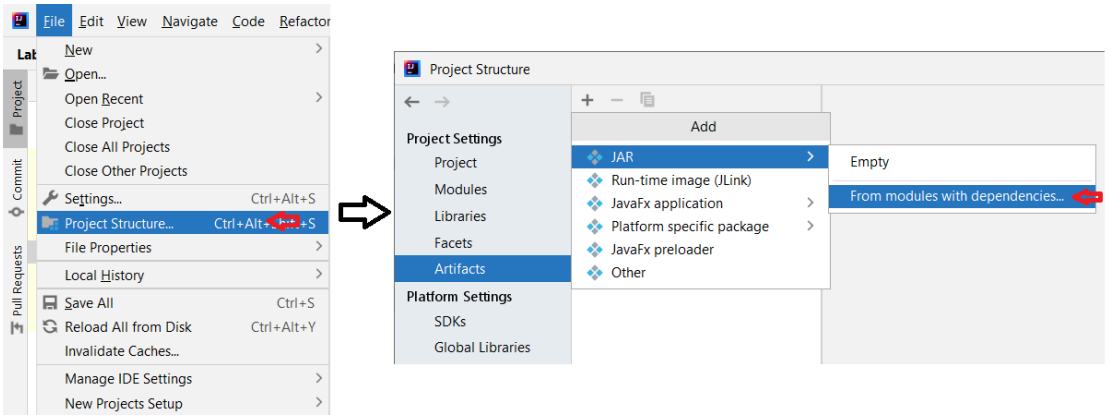
```
java -jar Server.jar 5000
java -jar Client.jar localhost 5000 1 demousername
```

Δημιουργία Executable Jar (IntelliJ):

Βήμα 1ο:

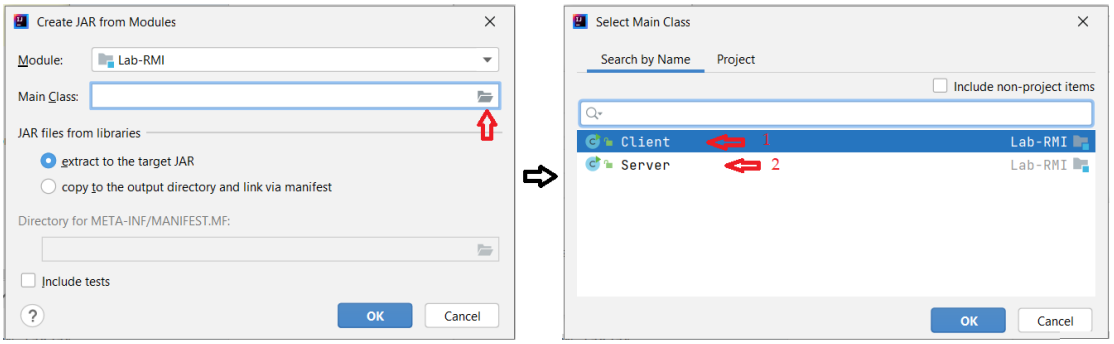
Ακολουθουμε τη διαδρομή…

File > Project Structure > Project Settings > Artifacts > Click green plus sign > Jar > From modules with dependencies…

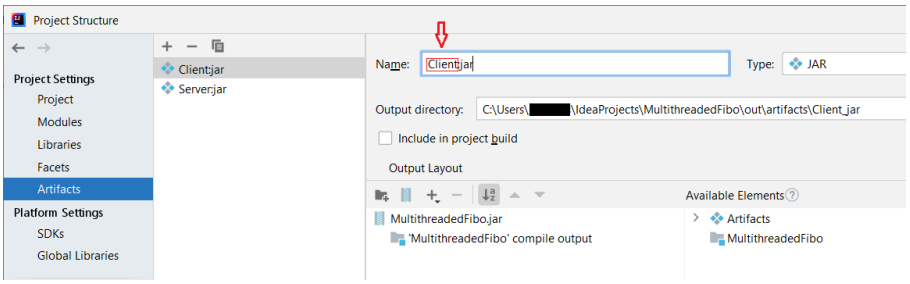


Βήμα 2ο:

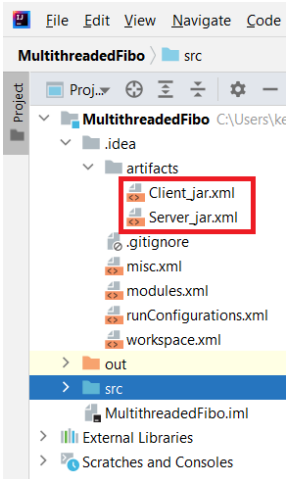
Έστω ότι έχουμε ένα project με δύο αρχεία που φέρουν τη μέθοδο main, αυτό σημαίνει ότι θα πρέπει να δημιουργήσουμε δύο .jar αρχεία. Στη δεύτερη εικόνα πιο κάτω εμφανίζονται τα δύο αυτά αρχεία του project πχ Client.java και Server.java. Τα βελάκια 1 και 2 υπονοούν ότι θα πρέπει να επαναληφθεί η διαδικασία δημιουργίας των αρχείων .jar και για τα δύο.



Για κάθε διαδικασία δημιουργίας αρχείου .jar θα πρέπει να δίνουμε το κατάλληλο τίτλο στο αρχείο που δημιουργούμε (π.χ. Client ή Server) μέσα στο κόκκινο πλαίσιο που δείχνει η παρακάτω εικόνα, καθ' ότι εξορισμού, το Intellij δίνει και στα δύο τον ίδιο τίτλο του project. Αυτό το κάνουμε για να ξέρουμε στο τέλος σε ποιους φακέλους θα παραχθούν τα τελικά .jar αρχεία για το κάθε ένα.

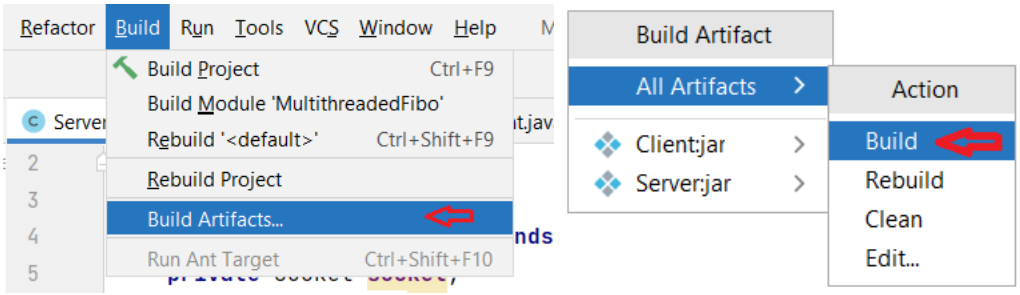


Έπειτα θα εμφανιστούν στον πλοηγό του project δύο xml αρχεία όπως φαίνεται στην παρακάτω εικόνα στο κόκκινο πλαίσιο:



Βήμα 3ο:

Τέλος, δημιουργούμε τα εκτελέσιμα συμπιεσμένα .jar όπως δείχνει η παρακάτω εικόνα.



Το Intellij θα αποθηκεύσει τα τελικά δύο .jar στο φάκελο out του project μέσα σε δύο υπο-φακέλους Client_jar και Server_jar, αντίστοιχα. Το Intellij αποδίδει το ίδιο όνομα (το όνομα του project δλδ <projectName>.jar) και στα δύο. Ωστόσο επειδή θα πρέπει να παραδοθούν σε έναν φάκελο με τίτλο jars -οπως ζητάει στα παραδοτέα, θα πρέπει να τα πάρετε και να τα μετονομάσετε (right-click > refactor > rename > (proceed anyway)) σε Client.jar και Server.jar όταν θα τα αποθηκεύσετε στο παραδοτέο φάκελο που θα γίνει τελικά ένα zip αρχείο.