Κεφάλαιο 13

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Σύνοψη

Σε αυτό το κεφάλαιο παρουσιάζεται αναλυτικά μία εργαστηριακή άσκηση προς εκπόνηση από τους σπουδαστές του βιβλίου. Πιο συγκεκριμένα, δίνεται ο κώδικας για την ανάπτυξη ενός πολυ-νηματικού εξυπηρετητή εφαρμογών, καθώς και οι οδηγίες για την εγκατάστασή του και το «τρέξιμό» του.

Προαπαιτούμενη Γνώση

1) Love, R., Are, S. H. W., Linus, A. C., Kernels, L. V. C. U., & Begin, B. W. (2005). *Linux kernel development second edition*. Novell Press.

2) Kerrisk, M. (2010). The Linux programming interface: a Linux and UNIX system programming handbook. No Starch Press.

3) Θραμπουλίδης, Κ. Χ. (2004). Αντικειμενοστρεφής προγραμματισμός – Java, ISBN: 9789608050808, Εκδότης: Τζιόλα.

4) Tanenbaum, A. S., & Van Steen, M. (2007). Distributed systems: principles and paradigms. Prentice-Hall.

5) Κάβουρας Ι.Κ., Μήλης Ι.Ζ., Ρουκουνάκη Α.Α., Ξηλωμένος Γ.Β. (2011), Κατανεμημένα συστήματα σε Java, 3η έκδοση, Εκδόσεις Κλειδάριθμος ΕΠΕ.

13.1 Απαιτούμενα

Για να πραγματοποιήσετε αυτό το εργαστήριο θα πρέπει να έχετε τα ακόλουθα εργαλεία:

- Το πιο πρόσφατο JDK (Java Developer Kit) που μπορείτε να κατεβάσετε δωρεάν από τη σελίδα: https://www.oracle.com/java/technologies/javase-jdk16-downloads.html
- Τη σουίτα NetBeans την οποία μπορείτε να κατεβάσετε δωρεάν από τη σελίδα: http://www.netbeans.org/downloads/

Σε περίπτωση που κάποιο από τα παραπάνω δεν λειτουργούν και τα δύο είναι διαθέσιμα από τη σελίδα της Oracle: https://www.oracle.com/gr/index.html.

13.2 Οδηγίες εγκατάστασης Java JDK και NetBeans

Αρχικά θα πρέπει να εγκαταστήσουμε το JDK και στη συνέχεια τη σουίτα NetBeans:

- Τρέχουμε την εγκατάσταση του JDK κάνοντας διπλό κλικ. Αν έχετε Windows 10 ή Windows 11 πατάτε Yes στο παράθυρο του UAC (User Access Control), αν το έχετε ενεργοποιημένο. Στη συνέχεια επιλέγετε Next στο Welcome παράθυρο που θα εμφανιστεί. Ύστερα θα σας εμφανιστεί το παράθυρο με τον φάκελο εγκατάστασης όπου θα εγκατασταθεί το JDK. Πατήστε Next χωρίς να πειράξετε τις ρυθμίσεις. Η εγκατάσταση ξεκινά. Πατήστε Next για να συνεχίσετε. Όταν η εγκατάσταση ολοκληρωθεί πατήστε Finish. Τέλος, δεν είναι υποχρεωτικό να κάνετε εγγραφή (register) στο JDK.
- Τρέχουμε την εγκατάσταση του NetBeans κάνοντας διπλό κλικ. Αν έχετε Windows 10 ή Windows 11 πατάτε Yes στο παράθυρο του UAC (αν έχετε ενεργοποιημένο το UAC). Αν έχετε κάποιο τείχος προστασίας (firewall) πιθανότατα να σας εμφανιστεί

ένα μήνυμα. Επιτρέψτε την πρόσβαση πατώντας «Allow access». Ύστερα θα σας εμφανιστεί το παράθυρο με τα εργαλεία που θα εγκαταστήσουν το NetBeans στον υπολογιστή σας. Πατήστε Next. Στη συνέχεια ελέγξτε το check box δίπλα από το οποίο λέει: "I accept the terms in the license agreement" και πατήστε Next. Ύστερα, στη φόρμα που εμφανίζεται επιλέξτε το πού θα εγκατασταθεί το NetBeans και πού έχετε εγκαταστήσει το JDK. Όπως και με την εγκατάσταση του JDK, απλώς πατήστε Next χωρίς να πειράξετε τίποτα. Συνεχίστε πατώντας install, ολοκληρώνοντας έτσι την διαδικασία της εγκατάστασης

13.3 Θεωρητικό Υπόβαθρο

Αναγκαίες θεωρούνται βασικές γνώσεις Java και αντικειμενοστραφούς προγραμματισμού (Object Oriented Programming). Επίσης, καλό είναι να έχετε μελετήσει τα κεφάλαια 2, 3 και 4 του βιβλίου αυτού. Παρακάτω παραθέτουμε περιεκτικά γνώσεις από αυτά τα κεφάλαια καθώς μερικά εισαγωγικά από τις πιο σύγχρονες τεχνικές προγραμματισμού που χρησιμοποιούνται σε αυτό το εργαστήριο.

13.3.1 Δικτύωση στη Java

13.3.1.1 Δημιουργία ενός Server

Υπάρχουν 5 βήματα για να δημιουργήσουμε έναν απλό εξυπηρετητή (server) στη Java:

1. Δημιουργία ενός `ServerSocket` αντικειμένου. Μια κλήση στον

`ServerSocket` constructor:

```
ServerSocket server
      = new ServerSocket(portNmbr, queueLength);
```

καταχωρεί ένα διαθέσιμο αριθμό πόρτας TCP (το οποίο θα χρησιμοποιούν οι Clients για να συνδεθούν - handshake point) και καθορίζει τον μέγιστο αριθμό Clients που μπορούν να περιμένουν στην ουρά ώστε να συνδεθούν με τον Server.

2. Κάντε το διακομιστή να ακούει επ ' αόριστον για μια προσπάθεια από έναν πελάτη για να συνδεθεί. Για να ακούσει για μια σύνδεση πελάτη ένα πρόγραμμα σε Java χρησιμοποιεί την μέθοδο accept της κλάσης ServerSocket:

```
Socket connection = Server.accept();
```

η οποία επιστρέφει ένα Socket όταν δημιουργηθεί η σύνδεση. Το Socket επιτρέπει στον server να αλληλεπιδράσει με τον πελάτη (σε μια διαφορετική πόρτα του server). Ένας multithreaded server μπορεί να πάρει το Socket που επιστράφηκε από κάθε κλήση για να αποδεχθεί και να δημιουργήσει ένα νέο thread, το οποίο θα διαχειρίζεται το network I/O με αυτό το Socket.

3. Στη φάση αυτή παίρνουμε τα αντικείμενα `OutputStream` και `InputStream` που δίνουν τη δυνατότητα στον server να επικοινωνήσει με τον client στέλνοντας και λαμβάνοντας bytes. Μπορούμε να χρησιμοποιήσουμε την τεχνική:

```
ObjectInputStream input =
```

```
        new ObjectInputStream(connection.getInputStream());

ObjectOutputStream output =
        new ObjectOutputStream(connection.getOutputStream());
```

για να «τυλίξουμε» stream types σε ένα `OutputStream` και `InputStream` που σχετίζονται με το Socket. Αν και συχνά είναι εύχρηστο να στείλουμε και να λάβουμε τιμές από `primitive types`, δηλ. αντικείμενα `Serializable`, από το να στέλνουμε και να λαμβάνουμε bytes.

Η κλήση της μεθόδου `flush` της `ObjectOutputStream` προκαλεί το `ObjectOutputStream` να στείλει ένα `stream header` στο αντίστοιχο `ObjectInputStream`.

Όταν χρησιμοποιείτε `ObjectOutputStream` και `ObjectInputStream` για να στείλετε και να λάβετε δεδομένα πάνω από μια σύνδεση του δικτύου, πάντα να δημιουργείτε το `ObjectOutputStream` πρώτο και κάντε `flush` το `stream` ώστε το αντίστοιχο `ObjectInputStream` να προετοιμαστεί για τη λήψη δεδομένων.

4. Ακολουθεί η φάση επεξεργασίας, κατά την οποία ο εξυπηρετητής (server) και ο πελάτης (client) επικοινωνούν μέσω των `OutputStream` και `InputStream` αντικειμένων.

5. Η επικοινωνία έχει ολοκληρωθεί, και ο server κλείνει την σύνδεση καλώντας την μέθοδο `close` στα `streams` του `socket`.

13.3.1.2 Δημιουργία ενός Client

Υπάρχουν τέσσερα (4) βήματα για να δημιουργήσουμε έναν απλό Client στην Java:

1. Δημιουργούμε ένα `Socket` για να συνδεθούμε στον server. Ο `Socket constructor` εγκαθιστά την σύνδεση με τον server:

```
Socket connection = new Socket (serverAddr, port);
```

2. Δημιουργούμε τα Socket methods του πελάτη, `getOutputStream` and `getInputStream` για να αποκτήσει αναφορές στα `OutputStream` και `InputStream` του Socket.

3. Ακολουθεί η φάση επεξεργασίας, κατά την οποία ο server και ο client επικοινωνούν μέσω των `OutputStream` και `InputStream` αντικειμένων.

4. Ο client κλείνει τη σύνδεση, όταν η επικοινωνία έχει ολοκληρωθεί καλώντας την μέθοδο `close` στα `streams` του `socket`.

13.3.2 Java Threads

Το προτιμώμενο μέσο για τη δημιουργία πολυνηματικών (multithreaded) εφαρμογών στη Java είναι με τη χρήση της διεπαφής (`interface`) `Runnable`. Ένα αντικείμενο `Runnable` αντιπροσωπεύει μια "διεργασία" που μπορεί να εκτελεστεί ταυτόχρονα με άλλες διεργασίες. Η διεπαφή `Runnable` δηλώνει μια ενιαία μέθοδο,τη run, η οποία περιέχει τον κώδικα που θα πρέπει να εκτελέσει το αντικείμενο `Runnable` . Όταν εκτελείται ένα thread, δημιουργείται και ξεκινάει ένα `Runnable`, και το νήμα καλεί την μέθοδο run του Runnable αντικειμένου, η οποία εκτελείται στο νέο νήμα.

Μια κλάση υλοποιεί (`implements`) την `Runnable`, έτσι ώστε πολλαπλές διεργασίες να μπορούν να εκτελούνται ταυτόχρονα.

Συνιστάται να χρησιμοποιήσετε τη διεπαφή `Executor`, ώστε να διαχειριστεί αυτή την εκτέλεση των αντικειμένων `Runnable` , για εσάς. Ένα αντικείμενο `Executor` δημιουργεί και διαχειρίζεται μια ομάδα από threads που ονομάζεται thread pool για να εκτελέσει αντικείμενα `Runnable`. Η χρήση `Executor` έχει πολλά πλεονεκτήματα σε σχέση με το να δημιουργήσετε νήματα μόνοι σας. Οι `Executor` μπορούν να επαναχρησιμοποιήσουν ήδη δημιουργημένα threads ώστε να εξαλείψουν τα έξοδα δημιουργίας ενός νέου νήματος (από την αρχή) για κάθε διεργασία και να βελτιώσουν την επίδοση με τη βελτιστοποίηση του αριθμού των threads, εξασφαλίζοντας έτσι ότι ο επεξεργαστής μένει απασχολημένος, χωρίς να δημιουργήσουν τόσα πολλά threads ώστε η εφαρμογή να μείνει χωρίς πόρους.

Η διεπαφή `Executor` δηλώνει μια ενιαία μέθοδο που ονομάζεται execute η οποία αποδέχεται ως argument ένα `Runnable`. Ο `Executor` εκχωρεί σε κάθε `Runnable` που πέρασε στην μέθοδο execute ένα από τα διαθέσιμα thread από την thread pool. Εάν δεν υπάρχουν διαθέσιμα threads, ο `Executor` δημιουργεί ένα νέο νήμα ή περιμένει να καταστεί διαθέσιμο ένα νήμα και αποδίδει το νήμα αυτό στην `Runnable` που πέρασε στη μέθοδο `execute`.

Η `Interface ExecutorService` είναι μια διεπαφή που `extends` (επεκτείνει) και δηλώνει μια σειρά από άλλες μεθόδους για τη διαχείριση του κύκλου ζωής ενός `Executor`. Ένα αντικείμενο που υλοποιεί (`implements`) το περιβάλλον `ExecutorService` μπορεί να δημιουργηθεί με μεθόδους `static` που δηλώθηκαν στην κλάση `Executors`.

Η μέθοδος `newCachedThreadPool` της Executors χρησιμοποιείται για την απόκτηση ενός `ExecutorService` που δημιουργεί νέα threads, όπως αυτά απαιτούνται από την εφαρμογή.

Η μέθοδος `newCachedThreadPool` της κλάσης `Executors` χρησιμοποιείται για την απόκτηση ενός αντικειμένου `ExecutorService` που δημιουργεί νέα νήματα όταν χρειάζονται από την εφαρμογή. Αυτά τα νήματα χρησιμοποιούνται από το αντικείμενο της `ExecutorService` ώστε να εκτελεστούν τα αντικείμενα των `Runnable`.

Η μέθοδος execute επιστρέφει στιγμιαία μετά από κάθε κλήση - το πρόγραμμα δεν περιμένει να τελειώσει για κάθε νήμα για να τελειώσει την εκτέλεσή του.

Η μέθοδος `shutdown` της `ExecutorService` ειδοποιεί το αντικείμενο της `ExecutorService` να σταματήσει να δέχεται νέα καθήκοντα, αλλά συνεχίζει να εκτελεί νήματα τα οποία έχουν γίνει ήδη αποδεχτά. Όταν όλα τα νήματα που έχουν γίνει αποδεχτά ολοκληρώσουν την εκτέλεσή τους το αντικείμενο της `ExecutorService` "πεθαίνει".

Σημείωση: Δεν μπορούμε να προβλέψουμε τη σειρά με την οποία τα νήματα θα χρονοπρογραμματιστούν για να εκτελεστούν, ακόμα και αν γνωρίζουμε με ποια σειρά δημιουργήθηκαν και μπήκαν σε κατάσταση “`runnable`”.

13.4. Κώδικας Java

Αρχικά δημιουργούμε τον Server και ύστερα τον Client. Για να δούμε τη λειτουργία τους, εκτελούμε τον Server πρώτα και ύστερα τον Client (σε αντίθετη περίπτωση ο Client δεν μπορεί να εξυπηρετηθεί, καθώς δεν υπάρχει ο Server που να τον εξυπηρετεί).

13.4.1 Κατασκευή προγράμματος Server

Ανοίγουμε το NetBeans και επιλέγουμε File → New Project → επιλέγουμε Java with Grandle (ενναλακτικά μπορούμε να επιλέξουμε και με maven ή Ant) στις Categories και ύστερα Java Application στα Projects και πατάμε Next → Κάνουμε Activate την Java SE σε περίπτωση που δεν έχει ενεργοποιηθεί (εάν χρειάζεται, θα έχει εμφανιστεί αντίστοιχο μήνυμα) → Γράφουμε το επιθυμητό όνομα της εφαρμογής του Server μας στο πεδίο Project Name → και πατάμε Finish.

Μέσα στην μέθοδο main γράφουμε:

```
  public static void main(String[] args)
        {
         Server application = new Server();
         application.execute();
        }
```

το οποίο δημιουργεί ένα αντικείμενο τύπου Server με το όνομα application και στη συνέχεια καλεί την μέθοδο execute αυτού του αντικειμένου.

Συνεχίζοντας, θα πρέπει να δημιουργήσουμε την κλάση Server και τη μέθοδο execute που ορίσαμε παραπάνω. Άρα ακολουθούμε τη διαδρομή File → New File → και επιλέγουμε Java στο Categories και Java Class στο File Types και πατάμε Next → Βάζουμε Server στο Class Name και πατάμε Finish.

Στη συνέχεια εισάγουμε τον παρακάτω κώδικα:

```
package server;

import java.io.EOFException;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;


public class Server
{
    private ServerSocket server;
    private Socket connection;

    private ExecutorService threadExecutor;


    public Server()
    {
        try
        {
```

```
            server = new ServerSocket(12345,100);
        }//end try
        catch(IOException ioException)
        {
            ioException.printStackTrace();
            System.exit(1);
        }// catch
        threadExecutor = Executors.newCachedThreadPool();
    }//end constructor

    public void execute()
    {
        try
        {
            while (true)
            {
                try
                {
                    waitForConnection();
                }//end try
                catch(EOFException eofException)
                {
                    System.out.println("Server terminated connec-
tion");
                }//end catch

            }//end while
        }//end try
        catch(IOException ioException)
        {
            ioException.printStackTrace();
        }//end try
    }//end execute

    private void waitForConnection() throws IOException
    {
        connection = server.accept();
        System.out.println("Connection recieved from "
                + connection.getInetAddress().getHostName());
        Threading call = new Threading(connection);
        threadExecutor.execute(call);
    }//end waitForConnection
    private class Threading implements Runnable
    {
        private ObjectInputStream input;
        private ObjectOutputStream output;
        private Socket connection;
        private int integer;

        public Threading(Socket socket)
        {
            connection = socket;
        }//end Constructor

        public void run()
        {
            try
            {
                getStreams();
                processConnection();
            }//end try
```

```
            catch(IOException ioException)
            {
                ioException.printStackTrace();
            }//end catch
            finally
            {
                closeConnection();
            }//end finally
        }//end run

        private void getStreams() throws IOException
        {
            output = new ObjectOutputStream(connec-
tion.getOutputStream());
            output.flush();
            input = new ObjectInputStream(connection.get-
InputStream());
            System.out.println("Got I/O Streams");
        }//end getStreams

        private void processConnection() throws IOException
        {
            System.out.println("Connection successfull");
            String message="Hallow";
            try
            {
                message = (String)input.readObject();
                System.out.println(message);
                integer = Integer.parseInt(message);
                integer += integer;
                output.writeObject(Integer.toString(integer));
                output.flush();
            }//end try

            catch(ClassNotFoundException classNotFoundException)
            {
                System.out.println("Unknown object type received");
            }//end catch
        }//end processConnection

        private void closeConnection()
        {
            try
            {
                output.close();
                input.close();
                connection.close();
            }//end try
            catch(IOException ioException)
            {
                ioException.printStackTrace();
            }//end catch
        }//end closeConnection

    }//end class Threads


}//end class Server
```

Με αυτόν τον τρόπο, έχει ολοκληρωθεί η δημιουργία του Server .

13.4.1 Κατασκευή προγράμματος Client

Για να δημιουργήσουμε το Client Project κάνουμε τις εξής ενέργειες:

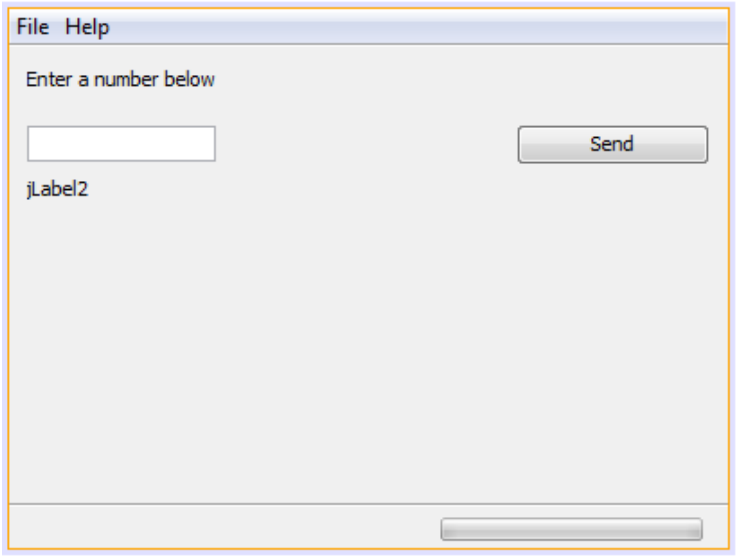
File → New project → Categories: Java with Grandle (εναλλακτικά με Maven ή Ant) → Projects: Java Application → Next → Name: Client→ Finish.

Στην συνέχεια, αφού δημιουργηθεί το παραπάνω Project στον φάκελο Source Packages κάνουμε δεξί click New →Other → Swing GUI Forms στα categories → JFrame Form στα File Types → Name Class (e.g. ClientView) → Finish.

Ενώ για να δημιουργήσουμε την γραφική διεπαφή με τον χρήση (GUI), εκτελούμε όσα περιγράφονται στα επόμενα.

Όπως παρατηρούμε, μας εμφανίζεται μια φόρμα κενή (που υπάρχει στο ClientView.Java) η οποία υλοποιείται με την έναρξη του προγράμματος. Στο Design του ClientView.java source file, κάνουμε drag & drop από την palette δύο Label (το πρώτο πάνω αριστερά και βάζουμε στα properties την τιμή Text: "Enter a number below" και το δεύτερο κάτω αριστερά), ένα Text Field (στην μέση αριστερά) και ένα Button (στην μέση δεξιά και βάζουμε στα properties στην τιμή Text: "Send"). Για να αρχίσουμε να γράφουμε τον κώδικά και να τον συνδέσουμε κατευθείαν με το Button κάνουμε απλώς διπλό κλικ στο Button. Στο action's method γράφουμε το όνομα της μεθόδου που θέλουμε να εκτελείται όταν ο χρήστης πατάει το κουμπί και μετά πατάμε το κουμπί «OK» – στη συγκεκριμένη περίπτωση βάζουμε sendButton. Τώρα μεταφερόμαστε στο Source του ClientView.java source file και ξεκινάμε να γράφουμε τη μέθοδο που μόλις δημιουργήσαμε.

Το παράθυρο που δημιουργείται θα είναι όπως φαίνεται στο σχήμα 13.1.



Σχήμα 13.1: Η γραφική διεπαφή του πελάτη (client) με τον χρήστη.

Προσθέτουμε τον ακόλουθο κώδικα, χωρίς να σβήσουμε τον προ-δημιουργημένο:

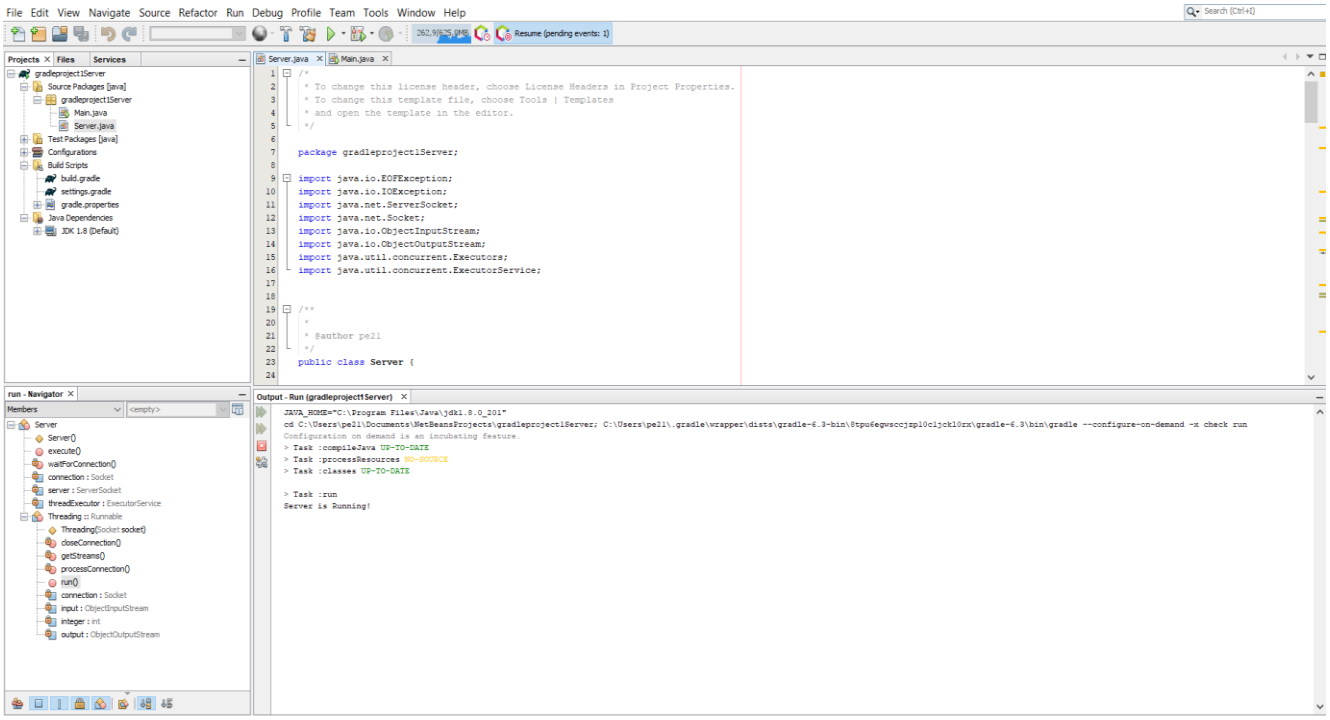
```
import java.net.Socket;
import java.net.InetAddress;
import java.io.IOException;
import java.io.EOFException;
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
```

```
public class ClientView extends FrameView
{

    private Socket client;
    private ObjectOutputStream output;
    private ObjectInputStream input;
    private int integer;
    private String message;

   public void sendButton()
    {
        integer = Integer.parseInt(this.jTextField1.getText());
        try
        {
            connectToServer();
            getStreams();
            processConnection();
        }//end try
        catch (EOFException eofException)
        {
            this.jLabel2.setText("Client terminated connection");
        }
        catch (IOException ioException)
        {
            this.jLabel2.setText("Client terminated connection");
        }//end catch
        finally
        {
            try
            {
                output.close();
                input.close();
                client.close();
            }//end try
            catch (IOException ioException)
            {
                this.jLabel2.setText("Closing connection");
            }//end catch
        }//end finally
    }//end sendButton
    private void connectToServer() throws IOException
    {
        client = new Socket(InetAddress.getLocalHost(),12345);
            this.jLabel2.setText("Connected to "
                    + client.getInetAddress().getHostName());
    }//end connectToServer
    private void getStreams() throws IOException
    {
            output = new ObjectOutputStream(cli-
ent.getOutputStream());
            output.flush();
            input = new ObjectInputStream(client.getInputStream());
            this.jLabel2.setText("Got I/O Streams");
    }//end getStreams
    private void processConnection() throws IOException
    {
        try
        {
```

```
            output.writeObject(Integer.toString(integer));
            output.flush();
            message = (String)input.readObject();
            this.jTextField1.setText(message);
        }//end try
        catch (ClassNotFoundException classNotFoundException)
        {
            this.jLabel2.setText("Unknown object type received");
        }//end catch
    }//end processConnection

}
```

Με αυτόν τον τρόπο, έχει ολοκληρωθεί η δημιουργία του πελάτη (client). Κάνουμε "Run" πρώτα τον Server και μετά τον Client για να δούμε την λειτουργία τους.

Στην σχήμα 13.2 απεικονίζεται η εκτέλεση του Server μετά την έναρξή του πατώντας το κουμπί "Run".



Σχήμα 13.2: Η εκτέλεση του Server μετά την έναρξή του στο Netbeans.

Βιβλιογραφικες Αναφορές

[1] Java Concurrency in Practice, by Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea

[2] Oaks, S. & Wong, H. (2004). *Java Threads*, 3rd edition. USA: O'Reilly Media

[3] Java Examples - Multithreaded Server, tutorial points, https://www.tutorialspoint.com/javaexamples/net_multisoc.htm.

[4] Savitch W. (2016), Απόλυτη Java, ISBN: 978-960-508-217-8, Εκδότης: Μ. Παρίκου & ΣΙΑ ΕΠΕ

Κριτήρια αξιολόγησης

Ερώτηση 1

Ποιος από τους παρακάτω αποτελεί τρόπο δημιουργίας ενός νήματος;
Α. Να κάνουμε extend την κλάση Concurrent
Β. Να υλοποιήοουμε το runnable interface και να δημιουργήσουμε το νήμα
Γ. Να υλοποιήοουμε το Serializable interface.

Ερώτηση 2

Σε μια επικοινωνία πελάτη-εξυπηρετητή χρειάζεται να ξεκινήσει πρώτα;
Α. Ο πελάτης
Β. Ο εξυπηρετητής
Γ. Αδιάφορο

Ερώτηση 3

Όταν ολοκληρωθεί η επικοινωνία πελάτη ή εξυπηρετητή χρειάζεται:
Α. Να κλείσουν τα sockets
Β. Να μπορούν να μείνουν ανοιχτά
Γ. Αδιάφορο

Ερώτηση 4

Όταν ολοκληρωθεί η επικοινωνία πελάτη - εξυπηρετητή χρειάζεται:
Α. Να κλείσουν τα αντίστοιχα sockets
Β. Να μπορούν να μείνουν ανοιχτά τα sockets που χρησιμοποιήθηκαν για την επικοινωνία
Γ. Αδιάφορο

Ερώτηση 5

Η χρήση της εντολής flush είναι σημαντική για να:
Α. Δημιουργήσουμε ένα socket
Β. Συγχρονίσουμε 2 νήματα
Γ. Αδειάσουμε το περιεχόμενο ενός socket και να ειδοποιήσουμε για την αποστολή του διαθέσιμου μηνύματος

Ερώτηση 6

Για να ακούσει μια σύνδεση πελάτη, ο διακομιστής χρησιμοποιεί την εντολή:
Α. listen()
Β. wait()
Γ. accept()