Κατανεμημένα Συστήματα

6ο Φυλλάδιο εργαστηρίου (27/04/2017)

Java Remote Method Invocation (RMI)

Η τεχνολογία Java Remote Method Invocation (RMI) αποτελεί άλλο ένα μοντέλο αντικειμενοστρεφούς σχεδιασμού με σκοπό να προάγει την διαλειτουργικότητα μεταξύ Java αντικειμένων σε ένα κατανεμημένο περιβάλλον με ετερογενή (heterogeneous) μηχανήματα. Αναπτύχθηκε από την Sun και συμπεριλήφθηκε στην έκδοση 1.1 του εργαλείου ανάπτυξης της Java (Java Development Kit). Η RMI αποτελεί ένα μηχανισμό που επιτρέπει τις κλήσεις μεθόδων μεταξύ αντικειμένων σε διαφορετικά εικονικά μηχανήματα Java. Από τη στιγμή που αποκτηθεί μία αναφορά σε ένα απομακρυσμένο αντικείμενο τότε αυτό μπορεί να χρησιμοποιηθεί σαν να ήταν τοπικό αντικείμενο. Η RMI εκτελεί όλες τις απαραίτητες λειτουργίες που αφορούν το απομακρυσμένο αντικείμενο με διαφανή τρόπο, ώστε να μην γίνονται αντιληπτές από τον προγραμματιστή, κάνοντας έτσι την ανάπτυξη κατανεμημένων Java εφαρμογών ιδιαίτερα εύκολη υπόθεση.

Η λειτουργική δομή του μοντέλου ακολουθεί την κλασική δομή της αρχιτεκτονικής πελάτη-εξυπηρετητή. Όταν ένας πελάτης επιθυμεί κάποιες λειτουργίες ενός αντικειμένου, στέλνει τα αιτήματα του προς τον εξυπηρετητή και δέχεται τα αποτελέσματα της επικοινωνίας. Μία βασική διαφορά από τα άλλα μοντέλα είναι ότι οποιοσδήποτε πελάτης είναι ένας Java πελάτης και οποιοδήποτε αντικείμενο εξυπηρετητής είναι ένα Java και μόνο Java αντικείμενο. Για τον λόγο αυτό το μοντέλο, το οποίο είναι αποκλειστικά Java-κεντρικό, μπορεί να εκμεταλλευτεί το ομοιογενές περιβάλλον της γλώσσας προγραμματισμού Java χωρίς την ανάγκη ύπαρξης κάποιας ιδιαίτερης γλώσσας ορισμού των διεπαφών και γενικότερα χωρίς τα προβλήματα που πηγάζουν από την επικοινωνία πολύ-γλωσσικών αντικειμένων.

Η αρχιτεκτονική ενός συστήματος Java RMI ακολουθεί την δομή των επιπέδων, όπως φαίνεται και στο ακόλουθο σχήμα. Ορίζονται τρία επίπεδα:

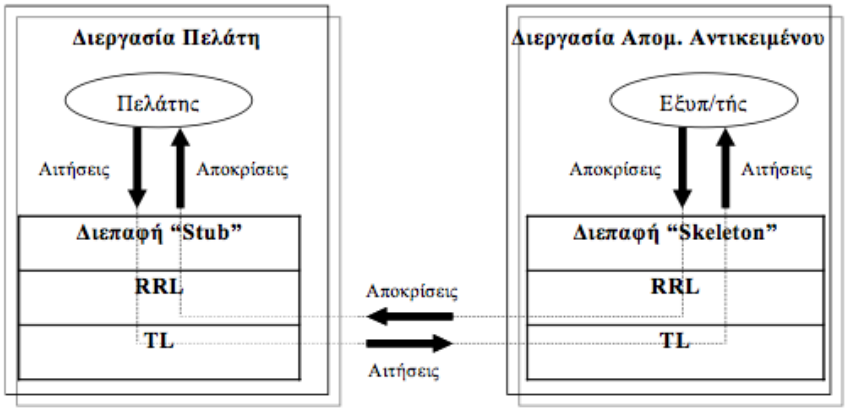
- το επίπεδο των διεπαφών stub / skeleton (Stub/Skeleton Layer - SSL)
- το επίπεδο απομακρυσμένης αναφοράς (Remote Reference Layer -RRL)
- το επίπεδο μεταφοράς (Transport Layer - TL).

Θα πρέπει να προσθέσουμε ότι υπάρχει άλλο ένα επίπεδο, το επίπεδο εφαρμογής (Application Layer), το οποίο δεν αποτελεί ένα καθαρό τμήμα ενός συστήματος RMI και βρίσκεται πάνω από τα τρία προαναφερθέντα.

Λειτουργία του συστήματος Java RMI

Ο μηχανισμός λειτουργίας του συστήματος RMI έχει ως εξής:

Ένας πελάτης, ο οποίος επιθυμεί κάποιες υπηρεσίες από ένα απομακρυσμένο αντικείμενο, στέλνει την αίτησή του η οποία προωθείται μέσα από τα επίπεδα του συστήματος προς τον κατάλληλο εξυπηρετητή του επιθυμητού αντικειμένου. Το σύστημα RMI είναι ο μηχανισμός που είναι υπεύθυνος να βρει και να προωθήσει την αίτηση του πελάτη. Αντιστρόφως, το απομακρυσμένο αντικείμενο περνάει την απόκριση μέσα στο σύστημα RMI το οποίο και δρομολογεί την απόκριση αυτή πίσω στον πελάτη. Στο σχήμα 2 απεικονίζεται η παραπάνω δρομολόγηση των αιτήσεων και των αποκρίσεων.



Σύμφωνα με το μοντέλο RMI ένας πελάτης ο οποίος επιθυμεί να εκτελέσει κάποιες υπηρεσίες ενός απομακρυσμένου αντικειμένου, πρέπει να καταθέσει μία αίτηση προς το σύστημα η οποία να περιέχει: την αναφορά προς το απομακρυσμένο αντικείμενο, τις επιθυμητές μεθόδους και τις απαραίτητες παραμέτρους για την υλοποίηση των μεθόδων αυτών. Όπως προαναφέραμε, το RMI είναι υπεύθυνο από εκεί και πέρα για την προώθηση της αιτήσεως. Η κατάθεση της αιτήσεως από τον πελάτη πραγματοποιείται μέσω της διεπαφής stub η οποία αποτελεί το ανώτερο επίπεδο του συστήματος και εμφανίζει το απομακρυσμένο αντικείμενο προς τον πελάτη σα να ήταν στην ίδια διεργασία. Η διεπαφή αυτή παρέχει επίσης στην αίτηση την κατάλληλη μορφοποίηση έτσι ώστε αυτή να μεταδοθεί στην τοποθεσία του απομακρυσμένου αντικειμένου. Μέσω της διεπαφής stub η αίτηση προωθείται στο επόμενο επίπεδο RRL. Το επίπεδο αυτό είναι

υπεύθυνο για τις τελευταίες προετοιμασίες της μετάδοσης. Επίσης, είναι αυτό που παρέχει στη διεπαφή stub τις παραμέτρους μορφοποίησης της αίτησης. Τέλος, η αίτηση κατεβαίνει στο τελευταίο επίπεδο TL το οποίο είναι υπεύθυνο για την μετάδοσή της. Στην τοποθεσία του εξυπηρετητή, το αντικείμενο συλλέγει την αίτηση με μία αντίστροφη διαδικασία, μέσω των επιπέδων του συστήματος RMI, και αποκρίνεται στον πελάτη με τρόπο παρόμοιο με τον δικό του.

Αναφέραμε ότι ο πελάτης, προκειμένου να καταθέσει την αίτησή του, θα πρέπει να έχει μία αναφορά προς το επιθυμητό αντικείμενο. Σε περίπτωση που αυτή η αναφορά δεν είναι γνωστή τότε το σύστημα RMI δίνει στον πελάτη την δυνατότητα να την αναζητήσει μέσω κατάλληλης υπηρεσίας που παρέχεται.

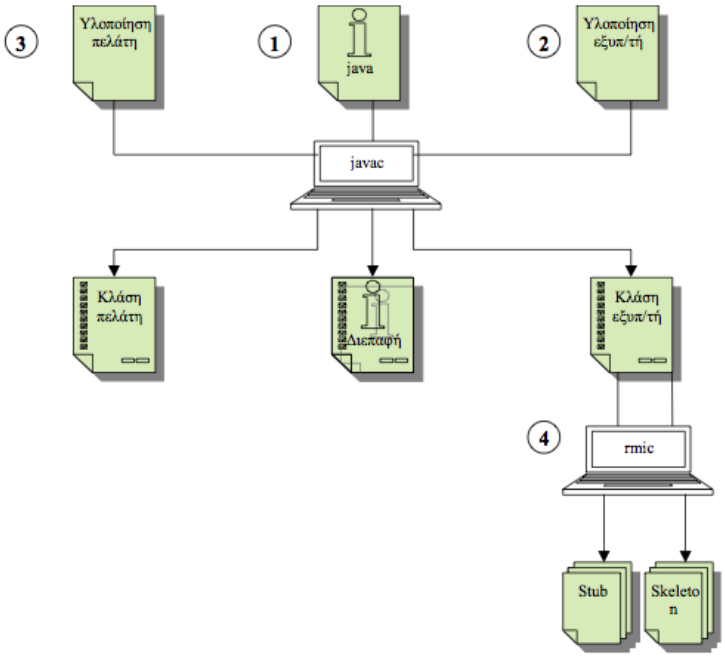
Σ' ένα σύστημα RMI ο πελάτης δεν έρχεται σε άμεση επαφή με το επιθυμητό αντικείμενο αλλά με την διεπαφή του αντικειμένου αυτού. Μέσω της διεπαφής εκτίθενται οι λειτουργίες τις οποίες μπορεί να αιτηθεί ένας πελάτης και τις οποίες υλοποιεί το απομακρυσμένο αντικείμενο.

Μοντέλο επικοινωνίας

Η επικοινωνία του πελάτη με το επιθυμητό αντικείμενο είναι η ίδια είτε το αντικείμενο βρίσκεται τοπικά είτε απομακρυσμένα. Το πρωτόκολλο επικοινωνίας που χρησιμοποιείται είναι το Java Remote Method Protocol (JRMP) το οποίο βασίζεται στο πρωτόκολλο επικοινωνίας TCP/IP.

Κατασκευή εφαρμογής πελάτη-εξυπηρετητή με Java RMI

Για την δημιουργία μίας εφαρμογής πελάτη-εξυπηρετητή βασισμένη στην τεχνολογία της Java RMI, η σειρά υλοποίησης που ακολουθείται είναι η κατασκευή της διεπαφής, της εφαρμογής του εξυπηρετητή και της εφαρμογής του πελάτη.



Η δημιουργία μίας εφαρμογής πελάτη-εξυπηρετητή στηριζόμενη στο μοντέλο Java RMI απαιτεί τις εξής προϋποθέσεις:

- Η γλώσσα προγραμματισμού που θα χρησιμοποιηθεί πρέπει να είναι η Java και μόνο αυτή.
- Τόσο στον υπολογιστή όπου θα τρέχει το πρόγραμμα εξυπηρετητή όσο και στον υπολογιστή απ' όπου θα τρέξει το πρόγραμμα πελάτη, θα πρέπει να είναι εγκατεστημένη η αντίστοιχη με τη γλώσσα, έκδοση του πακέτου ανάπτυξης της Java.

Κατασκευή της διεπαφής

Εδώ ο ρόλος της διεπαφής είναι η παρουσίαση προς τους πελάτες, του συνόλου των μεθόδων που προσφέρονται από τον εξυπηρετητή. Στο παράδειγμά μας, η διεπαφή, την οποία δηλώνουμε ως Operations.java, έχει ως εξής:

```
1. // Operations.java
2. import java.rmi.*;
3. public interface Operations extends Remote {
4.     public void setNum(int x, int y) throws RemoteException;
5.     public int sum() throws RemoteException;
6. }
```

Στη γραμμή 2 εισάγουμε τα απαραίτητα πακέτα (βιβλιοθήκες) προκειμένου να χρησιμοποιήσουμε διάφορα έτοιμα αντικείμενα (κλάσεις (classes), μεθόδους (methods), εξαιρέσεις (exceptions)) τα οποία παρέχονται από την Java. Στο συγκεκριμένο παράδειγμα χρησιμοποιούμε την διεπαφή Remote και την εξαίρεση RemoteException. Αντί για την εντολή της γραμμής 2 και με την οποία εισάγουμε όλα τα αντικείμενα που

περιλαμβάνονται στο πακέτο java.rmi θα μπορούσαμε να εισάγουμε μόνο αυτά που χρειαζόμαστε με τις εντολές:

```
import java.rmi.Remote;
import java.rmi.RemoteException;
```

Στην συνέχεια δηλώνουμε την διεπαφή με το όνομα `Operations`. Θα πρέπει να τονίσουμε ότι το όνομα που δίνουμε στο πρόγραμμα πρέπει να είναι το ίδιο με το όνομα της βασικής κλάσης του προγράμματος, στη συγκεκριμένη περίπτωση με το όνομα της διεπαφής.

Η διεπαφή χαρακτηρίζεται ως public έτσι ώστε να είναι δυνατό να κληθεί από έναν πελάτη απ' όπου και αν βρίσκεται. Σε περίπτωση που δεν χαρακτηρισθεί ως public, ο κάθε πελάτης, προκειμένου να την επικαλεσθεί, θα πρέπει να περιλαμβάνεται στο ίδιο πακέτο εργασίας με την διεπαφή. Τέλος, η διεπαφή θα πρέπει να επεκτείνει (extends) την διεπαφή Remote προκειμένου να δείχνει ότι αποτελεί απομακρυσμένο αντικείμενο. Μέσα στην διεπαφή δηλώνονται οι μέθοδοι οι οποίες παρέχονται και υλοποιούνται από τον εξυπηρετητή. Στο παράδειγμά μας έχουμε τις μεθόδους *setNum*() και *sum*().

Σε ότι αφορά την πρώτη μέθοδο, χαρακτηρίζεται ως *void* διότι δεν επιστρέφει κάποιο αποτέλεσμα ενώ οι μεταβλητές *int x* και *int y* αποτελούν τα ορίσματα της μεθόδου. Όπως έχουμε αναφέρει, τα ορίσματα αποτελούν τις παραμέτρους οι οποίες πρέπει να δοθούν από τον πελάτη προκειμένου να εκτελεστεί η μέθοδος στο περιβάλλον του εξυπηρετητή. Η δεύτερη μέθοδος, χαρακτηρίζεται ως *int* το οποίο σημαίνει ότι η μέθοδος αυτή όταν ολοκληρωθεί θα επιστρέψει μία ακέραια τιμή ως αποτέλεσμα. Επίσης παρατηρούμε ότι δεν έχει κάποιο όρισμα και άρα ο πελάτης την καλεί χωρίς να χρειάζεται να δώσει κάποιες παραμέτρους. Καθώς, οι μέθοδοι αυτοί θα κληθούν απομακρυσμένα θα πρέπει να προσέξουμε ότι οι παράμετροι και οι τύποι επιστροφής είναι είτε πρωταρχικοί τύποι (primitives) είτε αντικείμενα που υλοποιούν το interface *Serializable*.

Θα πρέπει να επισημάνουμε ότι κάθε μέθοδος που δηλώνεται πρέπει να δηλώνει υποχρεωτικά ότι ενδέχεται να δημιουργήσει εξαίρεση τύπου *RemoteException*, για κάθε πιθανότητα σφάλματος στη κλήση της μεθόδου.

Κατασκευή του εξυπηρετητή

Όπως έχουμε αναφέρει, ο εξυπηρετητής είναι υπεύθυνος για την παροχή των αιτουμένων υπηρεσιών προς τον πελάτη. Είναι ο παραλήπτης της αίτησης του πελάτη και αυτός που αναθέτει στην κατάλληλη μέθοδο την υλοποίηση της αιτούμενης υπηρεσίας.

Η κατασκευή του εξυπηρετητή μπορεί να επιτευχθεί σε δύο βήματα:

1. Την κατασκευή της κλάσης (implementation class) η οποία θα υλοποιεί τουλάχιστον κάποιες από τις απομακρυσμένες διεπαφές.

2. Την κατασκευή της κλάσης (server class) η οποία θα περιέχει την βασική συνάρτηση main().

Προφανώς, είναι δυνατή η κατασκευή μίας και μόνο κλάσης η οποία θα ικανοποιεί τα παραπάνω.

Τα βασικά στοιχεία της κάθε κλάσης είναι τα ακόλουθα:

1. Κλάση Υλοποίησης (Implementation Class)
   Η κλάση υλοποίησης θα πρέπει:
   - να δηλώνει ότι υλοποιεί κάποια, τουλάχιστον, διεπαφή (γραμμή 4),
   - να προσδιορίζει τον δομητή (constructor) της κλάσης (γραμμές 7-9)
   - να παρέχει την υλοποίηση των μεθόδων που δηλώνονται στην διεπαφή (γραμμές 10-13 και 14-16).

2. Κλάση Εξυπηρετητή (Server Class)
   Η κλάση εξυπηρετητή θα πρέπει:
   - να δημιουργεί και να εγκαθιστά, εάν επιθυμούμε, έναν διαχειριστή ασφάλειας (security manager) (γραμμές 19-20),
   - να δημιουργεί στιγμιότυπα (instances) του απομακρυσμένου αντικειμένου (remote object), δηλαδή της κλάσης υλοποίησης (γραμμή 21)
   - να καταχωρεί στο registry το απομακρυσμένο αντικείμενο με μία ονομασία σε μορφή URL, έτσι ώστε να είναι δυνατή η αναζήτησή της από τον πελάτη (γραμμή 22).

Στο παράδειγμά μας δημιουργήσαμε μία μόνο κλάση, αντί για δύο, την οποία ονομάζουμε OperationServer και την οποία έχουμε αποθηκεύσει σε ένα αρχείο με την ονομασία OperationServer.java.

```
1. // OperationServer.java
2. import java.rmi.*;
3. import java.rmi.server.*;
4. public class OperationServer extends UnicastRemoteObject implements
Operations {
5.     private int first;
6.     private int second;
7.     public OperationServer () throws RemoteException {
8.     super();
9. }
10.    public void setNum(int x, int y) throws RemoteException {
11.    first=x;
12.    second=y;
13. }
14.    public int sum() throws RemoteException {
15.    return first+second;
16. }
17.    public static void main(String args[]) {
18.    try {
19.    // RMISecurityManager security = new
```

```
            RMISecurityManager();
20.    // System.setSecurityManager(security);
21.    OperationServer server=new OperationServer();
22.    Naming.rebind("//localhost/OperationServer", server);
23.    }
24.    catch (Exception e) {
25.          System.out.println("OperationServer err: " + e);
26.          System.exit(1);
27.    }
28. }
29. }
```

Στην εφαρμογή του εξυπηρετητή διακρίνουμε τα εξής βασικά στοιχεία:

Στη γραμμή 4 κατά τη δήλωση της κλάσης η οποία δηλώνει ότι υλοποιεί τις μεθόδους της διεπαφής (με την εντολή implements Operations), πρέπει να επεκτείνεται και η απομακρυσμένη κλάση UnicastRemoteObject η οποία βρίσκεται στο πακέτο java.rmi.server. Με την δήλωση αυτή δίνεται η δυνατότητα στη κλάση να δημιουργήσει αντικείμενα, στιγμιότυπα της κλάσης, τα οποία θα μπορούν να χρησιμοποιήσουν τις προδηλωμένες (default) υποδοχές (sockets) του μοντέλου RMI για επικοινωνία. Επίσης, η δήλωση αυτή δίνει την δυνατότητα στην εφαρμογή του εξυπηρετητή να τρέχει συνεχώς περιμένοντας από κάποιον πελάτη την χρήση κάποιων υπηρεσιών. Το μοντέλο RMI δίνει την δυνατότητα να αναπτυχθούν και άλλες υποδοχές επικοινωνίας, οι οποίες να βασίζονται σε κάποιο άλλο πρωτόκολλο μεταφοράς, καθώς επίσης και την δυνατότητα η εφαρμογή του εξυπηρετητή να μην χρειάζεται να είναι ενεργή συνέχεια αλλά να ξεκινά όταν κάποιος πελάτης αιτηθεί κάποιων υπηρεσιών οι οποίες προσφέρονται από τον συγκεκριμένο εξυπηρετητή.

Στην γραμμή 7 και κατά την δήλωση του δομητή της κλάσης πρέπει αυτός να έχει την δυνατότητα να επιστρέφει μηνύματα λάθους τα οποία οφείλονται στην απομακρυσμένη κλήση από κάποιον πελάτη. Αυτό πρέπει να δηλώνεται διότι ο κατασκευαστής είναι αυτός ο οποίος καλείται κατά την δημιουργία ενός αντικειμένου-στιγμιοτύπου της κλάσης. Αυτό το αντικείμενο εκτίθεται προκειμένου ένας απομακρυσμένος πελάτης να μπορεί να το αναζητήσει για κάποιες υπηρεσίες. Από την στιγμή λοιπόν που θα έχουμε μία απομακρυσμένη επικοινωνία, θα πρέπει να ελέγχουμε και πιθανά λάθη κατά την διάρκεια της απομακρυσμένης επικοινωνίας.

Στις γραμμές 10-13 και 14-16 υλοποιούνται οι μέθοδοι, οι οποίες εκτίθενται από την διεπαφή, σύμφωνα με τις αρχές της Java.

Στις γραμμές 19 και 20 παρουσιάζεται η δυνατότητα, που παρέχει η τεχνολογία RMI, στην ασφάλεια του περιβάλλοντος του εξυπηρετητή από πιθανά μη επιτρεπτή πρόσβαση. Με την υλοποίηση ενός διαχειριστή ασφάλειας είμαστε σε θέση να μην επιτρέπουμε σε οποιονδήποτε την χρήση του εξυπηρετητή μας, αλλά σε όσους ικανοποιούν τις πολιτικές ασφάλειας που εμείς μπορούμε να θέσουμε. Βέβαια υπάρχει η δυνατότητα να μην χρησιμοποιήσουμε κάποιον διαχειριστή ασφάλειας αλλά αυτή η κίνηση θέτει σε κίνδυνο το περιβάλλον του εξυπηρετητή.

Η τελευταία εργασία που πρέπει να κάνουμε στην εφαρμογή μας, προκειμένου αυτή να ικανοποιεί τις απαιτήσεις του μοντέλου RMI, είναι να εκθέσουμε ένα αντικείμενο το οποίο θα είναι αυτό το οποίο θα είναι διαθέσιμο προς αναζήτηση από τους υποψήφιους πελάτες για την κλήση επιθυμητών υπηρεσιών. Η δημιουργία του αντικειμένου αυτού γίνεται με την δήλωση της γραμμής 21 αλλά η δημόσια έκθεση του πραγματοποιείται με την εντολή της γραμμής 22 με την οποία δηλώνουμε σε ποια διεύθυνση βρίσκεται το αντικείμενο και περιμένει πιθανή κλήση αλλά και με ποιο αναφορικό όνομα (OperationServer) θα πρέπει να αναζητηθεί από τους πελάτες. Έτσι

ο κάθε πελάτης, σύμφωνα με τις αρχές του μοντέλου RMI, προκειμένου να αιτηθεί την κλήση κάποιων απομακρυσμένων υπηρεσιών θα πρέπει να γνωρίζει, τουλάχιστον, την διεύθυνση στην οποία βρίσκεται το αντικείμενο από το οποίο θα αιτηθεί την υπηρεσία.

Εκτέλεση της εφαρμογής

Μετά την ολοκλήρωση της δημιουργίας του κώδικα που αφορά τον εξυπηρετητή έχουμε δύο αρχεία της μορφής interface_name.java και server_name.java (στο παράδειγμα μας έχουμε αντίστοιχα τα Operations.java και OperationServer.java). Μεταγλωττίζουμε τα αρχεία αυτά με τη βοήθεια του μεταφραστή της Java, **javac**, με αποτέλεσμα την δημιουργία των αρχείων κλάσεων της μορφής interface_name.class και server_name.class (στο παράδειγμα μας Operations.class και OperationServer.class). Ακολούθως εφαρμόζουμε τον μεταφραστή **rmic** πάνω στο αρχείο κλάσης του εξυπηρετητή server_name.class (OperationServer.class) με αποτέλεσμα την δημιουργία των επιθυμητών τμημάτων server_name_stub.class και server_name_skeleton.class (OperationServer_Stub.class και OperationServer_Skel.class).

Τα τελευταία βήματα που έχουμε να κάνουμε προκειμένου να εκτελέσουμε την εφαρμογή μας είναι:

- Έναρξη λειτουργίας του καταχωρητή (registry) στον οποίο θα γίνεται η καταχώρηση του αναφορικού ονόματος του απομακρυσμένου αντικειμένου, έτσι ώστε ο κάθε υποψήφιος πελάτης να μπορεί να αποκτήσει την αναφορά (reference) προς το επιθυμητό απομακρυσμένο αντικείμενο. Η έναρξη της λειτουργίας καταχώρησης υλοποιείται με την εντολή **start rmiregistry**.

- Έναρξη της λειτουργίας της εφαρμογής του εξυπηρετητή με την εντολή java server_name. Με την έναρξη λειτουργίας του εξυπηρετητή πραγματοποιείται η καταχώρηση του αναφορικού ονόματος του απομακρυσμένου αντικειμένου, το οποίο και αναμένει για τυχόν κλήσεις από πιθανούς πελάτες.

1ο Εργαστηριακή Άσκηση – RMI Server

Υλοποιήστε εξυπηρετητή με την χρήση RMI τεχνολογίας που θα παρέχει υπηρεσίες διαχείρισης τηλεφωνικού καταλόγου. Συγκεκριμένα, υλοποιήστε μεθόδους για:

1) Αναζήτηση επαφής με βάση το όνομα

2) Εισαγωγή νέας επαφής, ορίζοντας το όνομα, τη διεύθυνση και το τηλέφωνο της επαφής. Αν το όνομα της επαφής υπάρχει ήδη, δεν θα πρέπει να επιτρέπεται η εισαγωγή στον τηλεφωνικό κατάλογο.

Αρχικά ο εξυπηρετητής θα διαβάζει τις υπάρχουσες εγγραφές του τηλεφωνικού καταλόγου από σχετικό αρχείο κειμένου και θα τις αποθηκεύει σε κατάλληλη δομή δεδομένων. Το αρχείο με τις επαφές έχει την ακόλουθη μορφή:

Όνομα:Διεύθυνση:Τηλέφωνο

Απάντηση

```
import java.rmi.*;

public interface DirectoryOperations extends Remote {

    public Contact searchNumber(String name) throws RemoteException;
    public boolean insertContact (String name, String address, String number) throws
RemoteException;
    }

--------------------------------------------------------------------------------

import java.io.Serializable;

public class Contact implements Serializable {
    private String name;
    private String address;
    private String number;

    public Contact (String name, String address, String number){
        this.name=name;
        this.address=address;
        this.number=number;
    }

    public String getName(){
        return this.name;
    }

    public String getNumber(){
        return this.number;
    }
}


--------------------------------------------------------------------------------

import java.io.*;;
import java.util.*;
import java.rmi.*;
import java.rmi.server.*;


public class Server extends UnicastRemoteObject implements DirectoryOperations {
    private ArrayList<Contact> directory;

    public Server () throws RemoteException {
        super();
        String newline;
        String str[];

        try {
              directory = new ArrayList<Contact>();
              BufferedReader instream = new BufferedReader(new
FileReader("phonedirectory.txt"));

              while( (newline =instream.readLine()) != null ){
```

```
                    str = newline.split(":");
                    directory.add(new Contact(str[0], str[1], str[2]));
                }

            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }


 public Contact searchNumber(String name)  throws RemoteException{
        Contact ret=null;
        for(int i=0; i<directory.size();i++)
            if(name.equals(directory.get(i).getName()))
                   ret=directory.get(i);

        return ret;
   }


  public boolean insertContact(String name, String address, String number) throws
RemoteException {

       for(int i=0; i<directory.size();i++)
           if(name.equals(directory.get(i).getName()))
                   return false;

       directory.add(new Contact(name, address, number));
       System.out.println("Inserting "+ name);
       return true;
   }
 }


import java.rmi.*;

public class MainServer {

    public static void main(String args[]) {
       try {
       // RMISecurityManager security = new RMISecurityManager();
       // System.setSecurityManager(security);
        Server server=new Server ();
        Naming.rebind("//localhost/PhoneDirectory", server);
        System.out.println("Server up and running....");
       }
        catch (Exception e) {
            System.out.println("PhoneDirectory err: " + e);
               System.exit(1);
            }
   }


   // 2η υλοποίηση της main για εκτέλεση από το Netbeans
   public static void main(String args[]) {
        try {
              Server server=new Server();

             //Απαιτείται import java.rmi.registry.Registry;
 Registry r = java.rmi.registry.LocateRegistry.createRegistry(1099);//1099 is the port
                                        number
```

```
                Naming.rebind("//localhost/PhoneDirectory ", server);
                 System.out.println("Server up and running....");
                 } catch (Exception e) {
                         System.out.println("Server not connected: " + e);
                 }
        }
}
```