ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Εργαστήριο Κατανεμημένων Συστημάτων

Σημειώσεις

Φεβρουάριος 2012

Περιεχόμενα

Σχήματα

1. Εισαγωγή

Ένα κατανεμημένο σύστημα είναι ένα σύνολο ετερογενών υπολογιστών οι οποίοι διασυνδέονται μέσω ενός δικτύου, με στόχο την από κοινού παροχή υπηρεσιών στους χρήστες τους. Η εξέλιξη των κατανεμημένων συστημάτων επηρεάζεται από την εξέλιξη της τεχνολογίας των υπολογιστών και των δικτύων, αλλά και από τις συνεχώς μεταβαλλόμενες ανάγκες των χρηστών τους.

Από τεχνολογική; πλευράς, οι ραγδαίες εξελίξεις στο χώρο των υπολογιστών έχουν επιτρέψει τη μαζική παραγωγή πολύ οικονομικών , μικροεπεξεργαστών, οι οποίοι αποτελούν τις Κεντρικές Μονάδες Επεξεργασίας, ΚΜΕς (Central Processíng Uníts, CPUs) των σύγχρονων μικροϋπολογιστών. Εδώ και αρκετά χρόνια οι περισσότεροι χρήστες έχουν το δικό τους προσωπικό υπολογιστή (personal computer) ή σταθμό εργασίας (workstatíon), χωρίς να χρειάζεται πια να μοιράζονται ένα μεγάλο υπολογιστικό σύστημα με άλλους χρήστες.

Σημαντικές τεχνολογικές εξελίξεις έχουν σημειωθεί, όμως, και στα δίκτυα υπολογιστών, τα οποία επιτρέπουν την οικονομική σύνδεση δεκάδων, εκατοντάδων ή και εκατομμυρίων συστημάτων με τέτοιον τρόπο, ώστε τα διασυνδεόμενα συστήματα να ανταλλάσσουν μεγάλο όγκο πληροφοριών σε μικρό χρονικό διάστημα. Έτσι, παρά το ότι κάθε χρήστης έχει το δικό του σταθμό εργασίας, ο σταθμός αυτός μπορεί να επικοινωνήσει εύκολα και γρήγορα με οποιονδήποτε άλλον.

Η δυνατότητα εύκολης και οικονομικής διασύνδεσης πολλών προσωπικών υπολογιστών μέσω τοπικών δικτύων και δικτύων ευρείας περιοχής έχει δώσει ώθηση στη σχεδίαση υπολογιστικών συστημάτων πολλών επεξεργαστών, τα οποία ονομάζονται κατανεμημένα συστήματα (distributed systems), σε αντιδιαστολή με τα παραδοσιακά συγκεντρωτικά συστήματα (centralized systems) ενός επεξεργαστή. Στην πράξη ο όρος κατανεμημένο σύστημα είναι πιο εξειδικευμένος: ονομάζουμε κατανεμημένο σύστημα μία συλλογή από γεωγραφικά ανεξάρτητες, αυτόνομες υπολογιστικές μονάδες, που επικοινωνούν και διαλειτουργούν μεταξύ τους με τέτοιο τρόπο που παρουσιάζεται στους χρήστες του σαν ένα ενιαίο και συνεκτικό υπολογιστικό σύστημα. Το ενδιαφέρον για τα συστήματα αυτά οφείλεται στο ότι μπορούν να κατασκευαστούν με βάση απλούς (συγκεντρωτικούς ή μη) υπολογιστές και στο ότι, εκτός από αυξημένη επίδοση, επιτρέπουν και τον καταμερισμό των πόρων των ανεξάρτητων συστημάτων από τα οποία αποτελούνται. Ο τρόπος με τον οποίο καθορίζεται η εν λόγω συνεργασία και διαλειτουργικότητα αποτελεί τον πυρήνα ανάπτυξης κατανεμημένων συστημάτων.

Για την ανάπτυξη τέτοιων συστημάτων στα πλαίσια του εργαστηρίου θα χρησιμοποιηθεί ως γλώσσα προγραμματισμού η Java λόγω του προσανατολισμού της σε διαδικτυακές εφαρμογές και της σημαντικής υποστήριξης που παρέχει στον κατανεμημένο προγραμματισμό είτε μέσω ενσωματωμένων ευκολιών είτε μέσω τυποποιημένων πακέτων.

2. Κατανεμημένα συστήματα με Java

2.1. Εισαγωγή στην Java

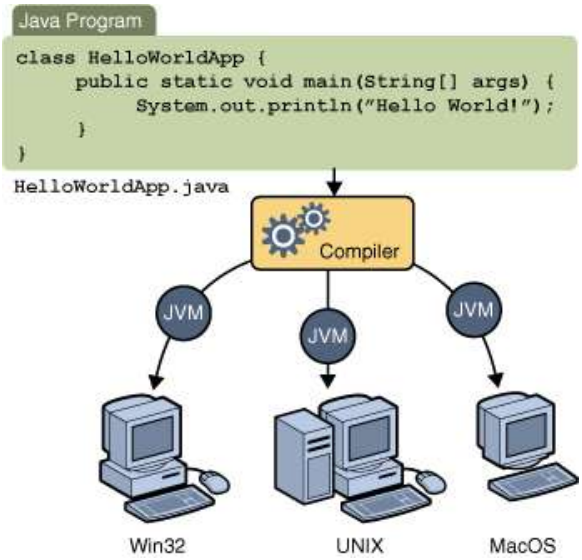
Η Java είναι γλώσσα προγραμματισμού και πλατφόρμα μαζί. Πρόκειται για υψηλού επιπέδου γλώσσα και μερικά από τα κύρια χαρακτηριστικά της είναι τα εξής :

- Απλή
- Αντικειμενοστραφής(object oriented)
- Κατανεμημένη (distributed**)**
- Πολυνηματική (multithreaded)
- Δυναμική
- Architecture neutral
- Μεταφέρσιμη (portable)
- Υψηλής απόδοσης
- Ανθεκτική
- Ασφαλής (secure)

Στην Java ολόκληρος ο πηγαίος κώδικας(source code) γράφεται σε αρχεία κειμένου με την κατάληξη `.java`. Έπειτα, τα πηγαία αρχεία γίνονται compile σε `.class` αρχεία μέσω του `javac` compiler. Ένα `.class` αρχείο δεν περιέχει κώδικα μεταφρασμένο στην γλώσσα μηχανής του συστήματος, αλλά bytecodes – κώδικα μεταφρασμένο σε γλώσσα Java εικονικής μηχανής (Java Virtual Machine). Έπειτα με το `java` εργαλείο τρέχει η εφαρμογή σε ένα στιγμιότυπο εικονικής μηχανής της Java.

Σχήμα 1: Επισκόπηση της διαδικασίας ανάπτυξης λογισμικού.

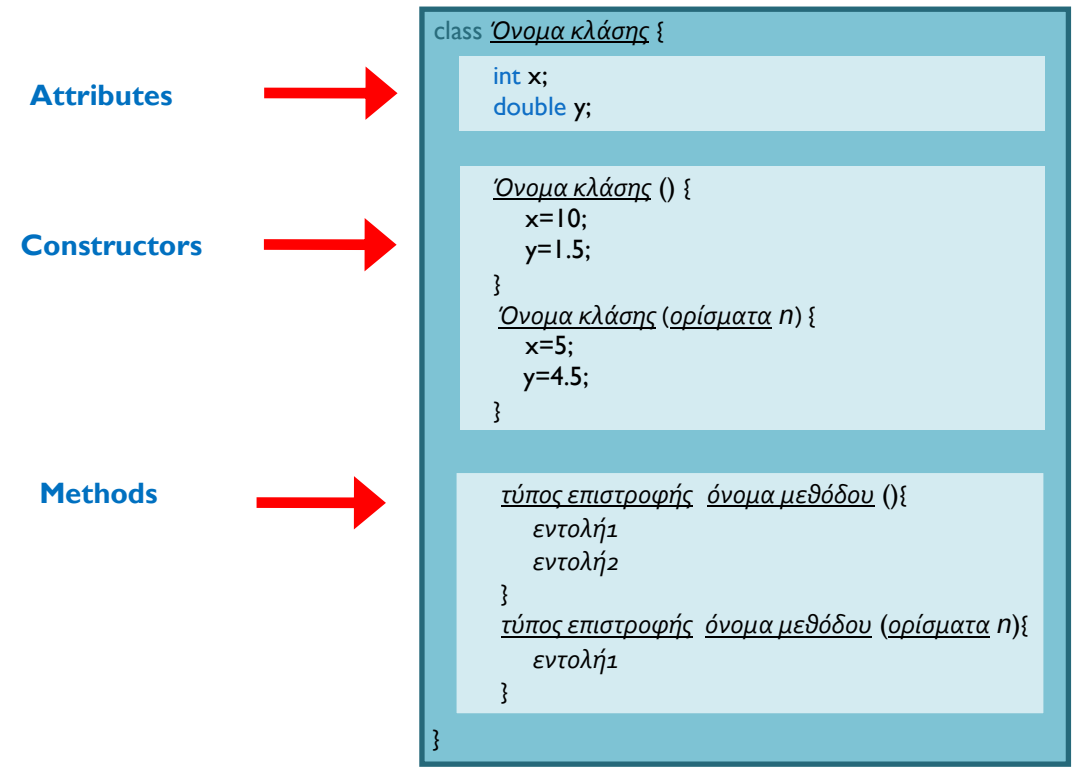
Για τον λόγο ότι η εικονική μηχανή της Java είναι διαθέσιμη για διάφορα λειτουργικά συστήματα, το ίδιο .class αρχείο μπορεί να τρέξει σε διαφορετικές πλατφόρμες(Microsoft Windows, Solaris OS, Linux, Mac OS).



Σχήμα 2: Λειτουργία της ίδιας εφαρμογής σε διαφορετικές πλατφόρμες.

2.2. Κλάσεις, αντικείμενα και στιγμιότυπα

Η **κλάση** (class) είναι ένα πρότυπο που χρησιμοποιείται για τη δημιουργία πολλαπλών αντικειμένων με παρόμοια χαρακτηριστικά. Οι κλάσεις περιέχουν όλα τα χαρακτηριστικά ενός συγκεκριμένου συνόλου αντικειμένων. Όταν γράφουμε ένα πρόγραμμα σε μια αντικειμενοστραφή γλώσσα προγραμματισμού, δεν ορίζουμε μεμονωμένα αντικείμενα αλλά κλάσεις αντικειμένων. Στον πραγματικό κόσμο υπάρχουν πολλά αντικείμενα από το ίδιο είδος. Για παράδειγμα το ποδήλατο κάποιου είναι ένα από τα πολλά ποδήλατα που κυκλοφορούν. Στον αντικειμενοστραφή προγραμματισμό το προαναφερθέν ποδήλατο είναι ένα αντικείμενο (instance - στιγμιότυπο) της κλάσης των ποδηλάτων. Γενικά τα ποδήλατα έχουν μερικά στοιχεία που χαρακτηρίζουν το είδος τους (π.χ. έχουν φρένα, έχουν ταχύτητες) και μερικά χαρακτηριστικά που είναι διαφορετικά σε κάθε ποδήλατο (π.χ. τρέχουσα ταχύτητα, τρέχων ρυθμός, μέγεθος ρόδας). Όταν ένα εργοστάσιο φτιάχνει ποδήλατα τα φτιάχνει με τα ίδια χαρακτηριστικά. Θα ήταν πολύ αναποτελεσματικό αν έφτιαχνε για κάθε ποδήλατο ένα καινούργιο σχέδιο. Στο συγκεκριμένο παράδειγμα μια κλάση είναι το σχέδιο από ένα είδος ποδηλάτου και τα αντικείμενα της είναι όλα τα παραγόμενα ποδήλατα από αυτό το σχέδιο. Τα ίδια ισχύουν και στον αντικειμενοστραφή προγραμματισμό. Ο προγραμματιστής έχει πλεονεκτήματα δημιουργώντας ένα προσχέδιο για αντικείμενα με ίδιες ιδιότητες. Μπορεί να λεχθεί ότι μια κλάση είναι το προσχέδιο ή το πρωτότυπο, το οποίο ορίζει κοινές μεταβλητές και κοινές μεθόδους (συναρτήσεις) για τα αντικείμενα του ίδιου είδους, ενώ ένα **αντικείμενο** (object) αναπαριστά μια ομάδα χαρακτηριστικών που εκτελούν κάποιες εργασίες και είναι αυτόνομο στοιχείο ενός προγράμματος. Η δομή μιας κλάσης φαίνεται στο παρακάτω σχήμα (Σχήμα 3).



Σχήμα 3: Δομή κλάσης στην Java.

Μία κλάση όπως βλέπουμε και από το Σχήμα 3 εκτός από μεταβλητές(attributes) και μεθόδους(methods) διαθέτει και κατασκευαστές(constructors) οι οποίοι καλούνται για να δημιουργήσουν αντικείμενα από τη κλάση-προσχέδιο. Η δήλωση κατασκευαστών μοιάζει με την δήλωση μεθόδων με την διαφορά ότι οι κατασκευαστές έχουν το ίδιο όνομα με την κλάση και δεν έχουν τιμή επιστροφής. Μία κλάση μπορεί να έχει περισσότερους του ενός κατασκευαστές αρκεί αυτοί να διαφέρουν στην δήλωση των ορισμάτων τους. Όπως και με τις μεθόδους η πλατφόρμα της Java διαφοροποιεί τους κατασκευαστές σύμφωνα με τον αριθμό και τον τύπο των ορισμάτων τους. Δεν γίνεται να υπάρχουν σε μία κλάση δύο κατασκευαστές που έχουν τον ίδιο αριθμό και τον ίδιο τύπο ορισμάτων στην ίδια σειρά, γιατί η πλατφόρμα δεν μπορεί να τους ξεχωρίσει. Κάνοντας κάτι τέτοιο θα προκύψει compile-time error.

Μια πιθανή υλοποίηση της κλάσης `Bicycle` που αναφέρθηκε προηγουμένως θα μπορούσε να είναι η εξής:

```
class Bicycle {

      //Attributes
      int speed = 0;
      int gear = 1;

      //Constructors
      Bicycle(){
           speed = 20;
           gear = 1;
      }

      Bicycle(int nSpeed, int nGear){
           speed = nSpeed;
           gear = nGear;
      }

      //Methods
      void changeGear(int newValue) {
           gear = newValue;
      }

      void speedUp(int increment) {
           speed = speed + increment;
      }

      void applyBrakes(int decrement) {
           speed = speed - decrement;
      }
}
```

Τα πεδία `speed`, και `gear` αναπαριστούν καταστάσεις στις οποίες βρίσκεται το αντικείμενο Bicycle, ενώ οι μέθοδοι(`changeGear`, `speedup`, κλπ) ορίζουν την αλληλεπίδραση του αντικειμένου με τον υπόλοιπο κόσμο.

Ωστόσο η κλάση `Bicycle` δεν έχει κάποια `main` μέθοδο. Αυτό γιατί δεν είναι ολοκληρωμένη εφαρμογή, αλλά απλά το προσχέδιο για ποδήλατα που μπορεί να χρησιμοποιηθεί σε μία εφαρμογή. Έτσι την ευθύνη για την δημιουργία και χρήση νέων ποδηλάτων(`new Bicycle`) μπορούν να την έχουν άλλες κλάσεις μίας εφαρμογής.

Για την δημιουργία ενός νέου ποδηλάτου, αρκεί να γράψουμε:

```
Bicycle myBike = new Bicycle();
```

όπου καλείται ο κατασκευαστής χωρίς ορίσματα της κλάσης `Bicycle` και δημιουργείται ένα νέο **στιγμιότυπο** (instance) της με όνομα `myBike`.

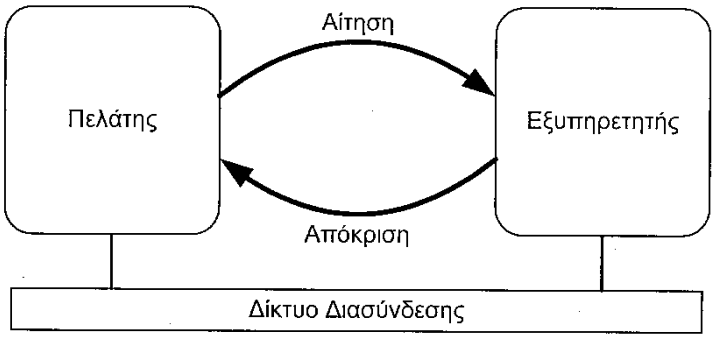
Η κλάση `BicycleDemo` παρακάτω δημιουργεί δύο ξεχωριστά αντικείμενα(instances - στιγμιότυπα) της κλάσης `Bicycle` και καλεί τις μεθόδους τους.

```
class BicycleDemo {
     public static void main(String[] args) {

          // Create two different Bicycle objects
          Bicycle bike1 = new Bicycle();
          Bicycle bike2 = new Bicycle(30, 0);

          // Invoke methods on those objects
          bike1.speedUp(10);
          bike1.changeGear(2);

          bike2.speedUp(10);
          bike2.changeGear(2);
          bike2.speedUp(10);
          bike2.changeGear(3);
          bike2.applyBrakes(10);
     }
}
```

2.3. Μοντέλο Client – Server

Εάν οι κόμβοι ενός κατανεμημένου συστήματος, δηλαδή οι μηχανές που συμμετέχουν σε αυτό, δεν έχουν κοινή μνήμη, οποιαδήποτε επικοινωνία μεταξύ τους πρέπει να βασίζεται σε ανταλλαγή μηνυμάτων μέσω του δικτύου. Όταν μία διεργασία-αποστολέας θέλει να επικοινωνήσει με μία διεργασία-παραλήπτη που εκτελείται σε μία διαφορετική μηχανή, τότε υλοποιείται επικοινωνία μεταξύ τους που βασίζεται σε ανταλλαγή μηνυμάτων μέσω ενός δικτύου.

Για να είναι αποτελεσματική η επικοινωνία, ο αποστολέας και ο παραλήπτης πρέπει να συμφωνούν στη σημασία των πληροφοριών που ανταλλάσσονται μεταξύ τους. Η συμφωνία αυτή πρέπει να λάβει χώρα σε διάφορα επίπεδα, από το κατώτερο, της μετάδοσης των *bits* μέσω των φυσικών συνδέσεων του δικτύου, έως το υψηλότερο, της ερμηνείας των μηνυμάτων από τις επικοινωνούσες εφαρμογές. Συνήθως θεωρούμε ότι η επικοινωνία μέχρι και το επίπεδο δικτύου παρέχεται από το πρωτόκολλο IP, ενώ σε ανώτερα επίπεδα χρησιμοποιούνται είτε τα γνωστά πρωτόκολλα του Διαδικτύου (TCP και UDP) είτε άλλα πρωτόκολλα.



Σχήμα 4: Μοντέλο πελάτη-εξυπηρετητή.

Στα κατανεμημένα λειτουργικά συστήματα χρησιμοποιούμε συνήθως μια λογική αφαίρεση που ονομάζεται μοντέλο πελάτη-εξυπηρετητή (client-server model), το οποίο απεικονίζεται στο Σχήμα 4. Στο μοντέλο αυτό το σύστημα δομείται ως ένα σύνολο διεργασιών, γνωστών ως εξυπηρετητών, οι οποίες προσφέρουν υπηρεσίες στις διεργασίες του χρήστη, γνωστές ως πελάτες. Οι πελάτες και οι εξυπηρετητές μπορεί να εκτελούνται στην ίδια ή σε διαφορετικές μηχανές, πάνω από το ίδιο ή διαφορετικά λειτουργικά συστήματα, αρκεί να υπάρχει συμφωνία μεταξύ τους στην αναπαράσταση και την ερμηνεία των μηνυμάτων που ανταλλάσσονται.

Η επικοινωνία ανάμεσα στον πελάτη και τον εξυπηρετητή χρησιμοποιεί ένα απλό πρωτόκολλο αίτησης/απάντησης (request/reply protocol) το οποίο, από λογικής άποψης, δεν είναι προσανατολισμένο σε σύνδεση. Στο πρωτόκολλο αυτό ο πελάτης στέλνει ένα μήνυμα αίτησης προς τον εξυπηρετητή, ζητώντας να του παρασχεθεί κάποια υπηρεσία. Ο εξυπηρετητής, που αναμένει συνεχώς αιτήσεις από τους πελάτες, παραλαμβάνει το μήνυμα αυτό, διεκπεραιώνει την αίτηση του πελάτη και του στέλνει ένα μήνυμα απάντησης. Το μήνυμα αυτό περιέχει τα δεδομένα που ζήτησε ο πελάτης ή την πληροφορία ότι η αίτησή του δεν ήταν δυνατό να διεκπεραιωθεί μαζί με κάποια αιτιολογία.

Μεταφράζοντας το μοντέλο αυτό σε Java θα μπορούσαμε αρχικά να δημιουργήσουμε μία κλάση για τον πελάτη(client) και μία για τον εξυπηρετητή(server) όπως παρακάτω.

```
//Κλάση πελάτη
public class Client {

	//Κατασκευαστές
	public Client() {
		//...απαραίτητος κώδικας για επικοινωνία με εξυπηρετητή.
	}

	//Μέθοδοι
	public void receiveData(){
		//...απαραίτητος κώδικας για λήψη μηνυμάτων.
	}

	public void sendData(){
		//...απαραίτητος κώδικας για αποστολή μηνυμάτων.
	}

	//main μέθοδος
	public static void main(String[] args) {
		new Client();
	}
}
```

```
//Κλάση εξυπηρετητή
public class Server {

	//Κατασκευαστές
	public Server() {
		//...απαραίτητος κώδικας για επικοινωνία με πελάτες.
	}

	//Μέθοδοι
	public void receiveData(){
		//...απαραίτητος κώδικας για λήψη μηνυμάτων.
	}

	public void sendData(){
		//...απαραίτητος κώδικας για αποστολή μηνυμάτων.
	}

	//main μέθοδος
	public static void main(String[] args) {
		new Server();
	}
}
```

Έπειτα, για την επικοινωνία των πελατών με τον εξυπηρετητή ανάλογα και το πρωτόκολλο επικοινωνίας που θέλουμε να χρησιμοποιήσουμε μπορούμε να χρησιμοποιήσουμε τις έτοιμες κλάσεις της βιβλιοθήκης της Java οι οποίες απλοποιούν την όλη διαδικασία.

2.4. Υποδοχές επικοινωνίας(Socket)

Γενικά, για να επικοινωνήσουν δύο διεργασίες μέσω του δικτύου, χρησιμοποιούν ένα πρωτόκολλο επιπέδου μεταφοράς (transport layer protocol), το οποίο παρέχει ένα λογικό κανάλι επικοινωνίας μεταξύ διεργασιών σε διαφορετικές μηχανές. Στο μοντέλο αναφοράς TCP/IP, το συγκεκριμένο κανάλι επικοινωνίας προσδιορίζεται πλήρως από πέντε στοιχεία: τις διευθύνσεις επιπέδου δικτύου των δύο άκρων, τις διευθύνσεις επιπέδου μεταφοράς των δύο άκρων και το πρωτόκολλο επικοινωνίας. Για παράδειγμα, στο TCP/IP οι διευθύνσεις επιπέδου δικτύου είναι οι διευθύνσεις IP, οι διευθύνσεις επιπέδου μεταφοράς είναι οι θύρες TCP ή UDP και το πρωτόκολλο μπορεί να είναι TCP ή UDP.

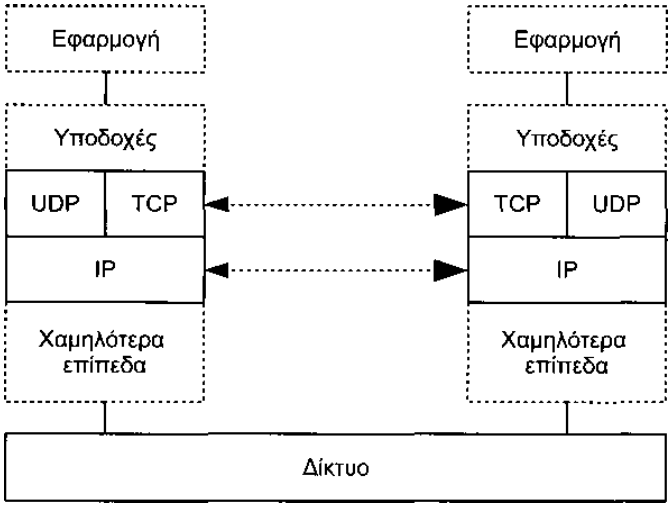
Υποδοχή (socket) ονομάζεται η λογική αφαίρεση ενός τέτοιου άκρου επικοινωνίας. Οι υποδοχές παρέχουν διεπαφή παρόμοια με αυτήν ενός περιγραφέα αρχείου, προκειμένου οι εφαρμογές να επικοινωνούν μέσω του δικτύου χρησιμοποιώντας τις συνηθισμένες κλήσεις εγγραφής/ανάγνωσης αρχείων, αντί για εξειδικευμένες κλήσεις επικοινωνίας μέσω του δικτύου. Οι υποδοχές προέρχονται από το Berkeley UNIX και χρησιμοποιούνται σε όλα σχεδόν τα λειτουργικά συστήματα.

Οι υποδοχές υποστηρίζουν τη χρήση πολλών πρωτοκόλλων μέσω της ίδιας διεπαφής. Η βιβλιοθήκη των υποδοχών μεταφράζει τις κλήσεις αυτές σε κατάλληλα μηνύματα του χρησιμοποιούμενου πρωτοκόλλου μεταφοράς, διαφανώς προς τον προγραμματιστή. Η μόνη διάκριση που γίνεται ανάμεσα στα πρωτόκολλα είναι ανάμεσα σε πρωτόκολλα ρευμάτων δεδομένων (stream protocols) και πρωτόκολλα δεδομενογραμμάτων (datagram protocols), τα οποία αντιστοιχούν σε διαφορετικούς τύπους υποδοχών. Οι υποδοχές ρευμάτων δεδομένων (stream sockets) υλοποιούν μετάδοση συνεχών ακολουθιών δυφιοσυλλαβών, για παράδειγμα μέσω του TCP, ενώ οι υποδοχές δεδομενογραμμάτων (datagram sockets) υλοποιούν μετάδοση αυτόνομων πακέτων, για παράδειγμα μέσω του UDP. Το πρωτόκολλο που χρησιμοποιείται δεν είναι ορατό στην εφαρμογή, παρά μόνο ο τύπος της υποδοχής.

Η βιβλιοθήκη που υλοποιεί τη διεπαφή των υποδοχών μοιάζει με ένα επίπεδο στη στοίβα πρωτοκόλλων αφού οι διεργασίες του χρήστη επικοινωνούν με αυτή τη βιβλιοθήκη, η οποία με τη σειρά της επικοινωνεί με τα πρωτόκολλα του επιπέδου μεταφοράς. Η συγκεκριμένη διάταξη για τα πρωτόκολλα του Διαδικτύου απεικονίζεται στο Σχήμα 5. Ωστόσο οι βιβλιοθήκες των υποδοχών δεν επικοινωνούν μεταξύ τους με κάποιο πρωτόκολλο. Στην πραγματικότητα δεν είναι καν απαραίτητο και τα δύο άκρα να χρησιμοποιούν τη διεπαφή των υποδοχών. Κάθε άκρο μπορεί να χρησιμοποιεί οποιαδήποτε διεπαφή θέλει, αρκεί να διασφαλίζεται η επικοινωνία μεταξύ των πρωτοκόλλων στο επίπεδο μεταφοράς.

Η διεπαφή των υποδοχών μπορεί να χρησιμοποιηθεί για την επικοινωνία είτε μεταξύ πελατών και εξυπηρετητών είτε μεταξύ ομοτίμων, σε συνδυασμό είτε με το πρωτόκολλο TCP είτε με το πρωτόκολλο UDP, ανάλογα με τις ανάγκες της εφαρμογής. Στο μοντέλο πελάτη-εξυπηρετητή, η βασική ροή ενός εξυπηρετητή είναι να λαμβάνει απαιτήσεις από τους πελάτες, να τις εκτελεί και, πιθανόν, να ενημερώνει τους πελάτες για τα αποτελέσματα. Γενικά, κάθε εξυπηρετητής παραμένει ενεργός για μεγάλα χρονικά διαστήματα για να εξυπηρετεί τις αιτήσεις πολλών πελατών. Αν

πολλοί πελάτες υποβάλλουν ταυτόχρονα αιτήσεις σε αυτόν, ο εξυπηρετητής μπορεί να τις χειριστεί είτε ταυτόχρονα είτε διαδοχικά τη μία μετά την άλλη.



Σχήμα 5: Υποδοχές και πρωτόκολλα του Διαδικτύου.

Ένας ταυτόχρονος εξυπηρετητής (concurrent server) επικοινωνεί ταυτόχρονα με πολλούς πελάτες, δηλαδή έχει τη δυνατότητα να διατηρεί ταυτόχρονα πολλά ενεργά κανάλια επικοινωνίας. Για να επιτευχθεί αυτό, κατά την άφιξη κάθε νέας αίτησης μέσω μιας υποδοχής δημιουργείται ένα αντίγραφο της υποδοχής αυτής. Με αυτόν τον τρόπο, ενώ ο εξυπηρετητής χειρίζεται την αίτηση χρησιμοποιώντας το αντίγραφο της υποδοχής, μπορεί να συνεχίσει να δέχεται αιτήσεις από την αρχική υποδοχή. Η ταυτόχρονη εξυπηρέτηση πολλών πελατών απαιτεί επιπλέον τη δημιουργία μιας νέας διεργασίας ή ενός νέου νήματος (βλ. Νήματα (Threads)) για κάθε αίτηση, με τη νέα διεργασία (ή το νέο νήμα) να χειρίζεται το αντίγραφο της αρχικής υποδοχής.

Αντίθετα, ένας επαναληπτικός εξυπηρετητής (iterative server) επικοινωνεί μόνο με έναν πελάτη κάθε στιγμή. Αυτό σημαίνει ότι δεν είναι απαραίτητη η δημιουργία αντιγράφων των υποδοχών επικοινωνίας του εξυπηρετητή αφού, αν δεν ολοκληρωθεί η επεξεργασία της τρέχουσας αίτησης, δεν είναι δυνατή η αποδοχή της επόμενης. Προφανώς, σε αυτή την περίπτωση μόνο μία διεργασία (ή μόνο ένα νήμα) επαρκεί για την εξυπηρέτηση όλων των αιτήσεων. Όμως, στον επαναληπτικό εξυπηρετητή απαιτείται η δυνατότητα προσθήκης των αιτήσεων που καταφθάνουν όταν ο εξυπηρετητής είναι απασχολημένος σε μια ουρά (queue) του συστήματος, προκειμένου αυτές να εξυπηρετηθούν αργότερα.

Για να μπορούν οι πελάτες να επικοινωνήσουν με έναν εξυπηρετητή σε κάποια μηχανή, θα πρέπει να γνωρίζουν τη θύρα TCP ή UDP στην οποία αναμένει αιτήσεις ο εξυπηρετητής. Ορισμένοι κοινοί εξυπηρετητές χρησιμοποιούν πάντα μια ευρέως γνωστή θύρα (well known port). Για παράδειγμα, οι εξυπηρετητές του Παγκόσμιου Ιστού συνήθως χρησιμοποιούν τη θύρα 80 του TCP. Για να αποφύγουμε τη στατική εκχώρηση θυρών, μπορούμε να χρησιμοποιήσουμε μια κεντρική διεργασία στην οποία θα καταγράφονται από όλους τους ενεργούς εξυπηρετητές οι τρέχουσες θύρες που χρησιμοποιούν. Αυτή πρέπει να χρησιμοποιεί κάποια ευρέως γνωστή θύρα προκειμένου οι πελάτες να μπορούν να μάθουν την τρέχουσα θύρα του εξυπηρετητή που τους ενδιαφέρει.

Η βασική διαφοροποίηση του μοντέλου ομοτίμων από το μοντέλο πελάτη-εξυπηρετητή είναι ότι κάθε ομότιμος κόμβος λειτουργεί ταυτόχρονα και ως πελάτης και ως εξυπηρετητής. Γι' αυτό, οι ομότιμοι είναι οργανωμένοι σχεδόν πάντα ως ένα σύνολο ταυτόχρονων διεργασιών ή νημάτων (βλ. Νήματα (Threads)) που έχουν το ρόλο είτε του πελάτη είτε του εξυπηρετητή. Δηλαδή η οργάνωσή τους είναι παρόμοια με αυτή των ταυτόχρονων εξυπηρετητών στο μοντέλο πελάτη-εξυπηρετητή, με τη διαφοροποίηση ότι σε αυτή την περίπτωση και ο κώδικας του πελάτη εκτελείται ταυτόχρονα με αυτόν του εξυπηρετητή.

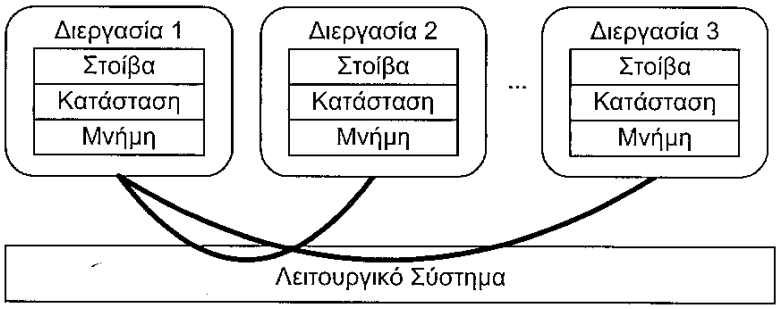
Η κλασική διεπαφή των υποδοχών έχει γραφτεί για τη γλώσσα C, και είναι τόσο περίπλοκη, ώστε να καλύπτει οποιοδήποτε πρωτόκολλο με τυποποιημένο τρόπο, παρά τις διαφορές μεταξύ των πρωτοκόλλων σε θέματα ονομασίας, διευθυνσιοδότησης και επικοινωνίας, καθώς και τις διαφορές μεταξύ των διαφόρων τύπων υποδοχών. Για παράδειγμα στην κλήση δημιουργίας μίας υποδοχής ο προγραμματιστής πρέπει να συμπληρώσει τα πεδία μιας γενική; δομής δεδομένων με συγκεκριμένες τιμές για να δείξει ότι η υποδοχή θα χρησιμοποιήσει τα πρωτόκολλα TCP/IP. Συμπληρώνοντας διαφορετικές τιμές στα πεδία αυτά ο προγραμματιστής μπορεί να χρησιμοποιήσει οποιοδήποτε πρωτόκολλο επιθυμεί, μέσω της ίδιας ακριβώς διεπαφής.

Επειδή η συντριπτική πλειοψηφία των εφαρμογών χρησιμοποιεί τα πρωτόκολλα TCP, UDP και IP, οι σύγχρονες γλώσσες προγραμματισμού, όπως η Java, απλοποιούν την κλασική διεπαφή των υποδοχών, ώστε να επιτυγχάνεται με απλούστερο τρόπο η εγκαθίδρυση επικοινωνίας. Συγκεκριμένα το τυποποιημένο πακέτο *java.net* περιέχει κλάσεις για τη δημιουργία της υποδομής ενός δικτύου σε Java. Οι κλάσεις αυτές καλύπτουν τις διευθύνσεις, τις υποδοχές, κλπ, όπως θα δούμε στην συνέχεια για κάθε περίπτωση χωριστά(UDP και TCP).

2.5. Νήματα (Threads)

Στα περισσότερα κλασικά συστήματα πολυπρογραμματισμού η κάθε διεργασία(process) έχει ένα χώρο διευθύνσεων και μια ροή ελέγχου, δηλαδή μία ακολουθία εντολών που εκτελεί σε κάποια χρονική στιγμή. Κάθε διεργασία εκτελείται σε έναν εικονικό επεξεργαστή. Οι εικονικοί επεξεργαστές μπορούν είτε να μοιράζονται ψευδοταυτόχρονα σε ένα μόνο επεξεργαστή είτε να καταχωρίζονται ταυτόχρονα στους διάφορους επεξεργαστές ενός συστήματος πολλών επεξεργαστών.

Στόχος του λειτουργικού συστήματος είναι να (κατα)μερίσει με διαφανή τρόπο τον επεξεργαστή ή τους επεξεργαστές και τους άλλους πόρους του συστήματος στους εικονικούς επεξεργαστές, αποτρέποντας τις διεργασίες από το να επηρεάσουν την ορθότητα εκτέλεσης των άλλων διεργασιών. Όπως φαίνεται στο Σχήμα 6, κάθε διεργασία απομονώνεται από τις υπόλοιπες για λόγους διαφάνειας και ασφάλειας, και η επικοινωνία μεταξύ των διεργασιών επιτρέπεται μόνο μέσω ειδικών μηχανισμών που παρέχει το λειτουργικό σύστημα. Για να επιτευχθεί η απομόνωση, απαιτούνται μηχανισμοί προστασίας.



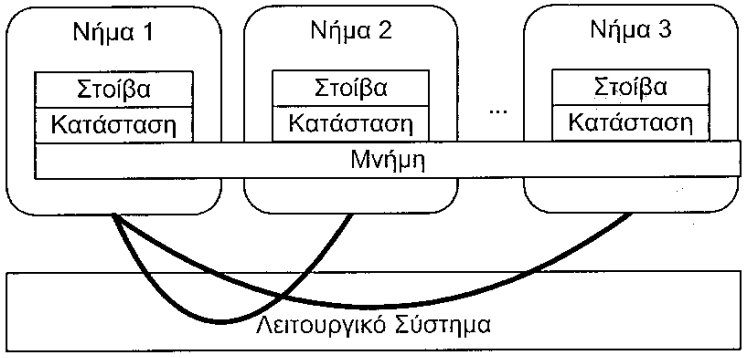
Σχήμα 6: Επικοινωνία διεργασιών.

Η υλοποίηση των διεργασιών έχει αρκετά υψηλό κόστος. Η δημιουργία μίας νέας διεργασίας απαιτεί τη δημιουργία ενός νέου χώρου εικονικών διευθύνσεων, για τον οποίο θα πρέπει να κατασκευαστεί ένας πίνακας σελίδων ή/και τμημάτων, και στον οποίο θα πρέπει να φορτωθεί ο κώδικας του προγράμματος και να αρχικοποιηθούν οι περιοχές δεδομένων, σωρού και στοίβας. Η μεταγωγή των συμφραζομένων των διεργασιών, πράγμα απαραίτητο για να έχουμε (κατα)μερισμό των πόρων ανάμεσα στους εικονικούς επεξεργαστές, απαιτεί την αποταμίευση και την αποκατάσταση περιβαλλόντων διεργασιών, δηλαδή, μεταξύ άλλων, αποθήκευση και φόρτωση των περιεχομένων των καταχωρητών του επεξεργαστή, τροποποίηση των καταχωρητών του διαχειριστή μνήμης, αλλαγή πίνακα σελίδων ή/και τμημάτων και ακύρωση των περιεχομένων όλων των κρυφών μνημών, καθώς και του ενταμιευτή μετάφρασης εικονικών διευθύνσεων.

Σε πολλές περιπτώσεις είναι επιθυμητό να έχουμε περισσότερες της μίας ροές ελέγχου στην ίδια εφαρμογή, οι οποίες να μοιράζονται τον ίδιο χώρο διευθύνσεων και να εκτελούνται φαινομενικά παράλληλα (*quasi-parallel*). Οι ροές αυτές είναι γνωστές ως νήματα ελέγχου (*threads of control*). ή απλώς νήματα, και μπορούν να θεωρηθούν ως υποδιεργασίες που μοιράζονται το χώρο διευθύνσεων μίας διεργασίας. Κάθε νήμα εκτελείται ακολουθιακά, έχει το δικό του μετρητή προγράμματος και τη δική του στοίβα. Όλα τα νήματα μίας διεργασίας μοιράζονται, όμως, τα ίδια τμήματα κώδικα

και δεδομένων, καθώς και τις ίδιες τιμές καταχωρητών για τη μονάδα διαχείρισης μνήμης, αφού αξιοποιούν τον ίδιο χώρο διευθύνσεων. Τα νήματα μίας διεργασίας μοιράζονται επίσης στοιχεία όπως τα ανοικτά αρχεία, τα χρονόμετρα και τα σήματα της διεργασίας.

Γενικά ένα νήμα μπορεί να κάνει ότι και μια διεργασία και να βρίσκεται σε παρόμοιες καταστάσεις με αυτήν. Τα διάφορα νήματα μπορούν να εκτελούνται ψευδοπαράλληλα στον ίδιο επεξεργαστή ή (πραγματικά) παράλληλα σε περισσότερους του ενός επεξεργαστές οι οποίοι, όμως, μοιράζονται την ίδια μνήμη. Τα νήματα μπορούν να δημιουργούν διεργασίες-παιδιά, να εκδίδουν κλήσεις του συστήματος, να εμποδίζονται περιμένοντας να συμβεί κάποιο γεγονός ή συμβάν κ.ο.κ. Όταν ένα νήμα εμποδιστεί για οποιονδήποτε λόγο, τότε μπορεί να εκτελεστεί είτε μία άλλη εκτελέσιμη διεργασία είτε ένα άλλο εκτελέσιμο νήμα της ίδιας διεργασίας, ανάλογα με την πολιτική χρονοπρογραμματισμού του λειτουργικού συστήματος. Όπως φαίνεται στο Σχήμα 7, τα νήματα μίας διεργασίας μπορούν να επικοινωνούν μεταξύ τους όχι μόνο μέσω των ειδικών μηχανισμών του λειτουργικού συστήματος, αλλά και απευθείας μέσω της κοινής μνήμης τους.



Σχήμα 7: Επικοινωνία νημάτων.

Αφού τα νήματα μίας διεργασίας μοιράζονται τον ίδιο χώρο διευθύνσεων, μπορούν να προσπελάσουν και να τροποποιήσουν οποιαδήποτε θέση του κοινού χώρου διευθύνσεων επιθυμούν. Ένα νήμα μπορεί, δηλαδή, να αλλάξει τις καθολικές μεταβλητές της διεργασίας, ακόμη και να διαγράψει τη στοίβα ενός άλλου νήματος. Δεν υπάρχει, ούτε και μπορεί να επιβληθεί, κάποιος μηχανισμός προστασίας μεταξύ των νημάτων μιας διεργασίας, αφού αντίθετα με τις διεργασίες των διαφόρων χρηστών που μπορεί να βρίσκονται σε συναγωνισμό μεταξύ τους, τα νήματα ανήκουν στην ίδια διεργασία χρήστη η οποία τα έχει δημιουργήσει για να συνεργάζονται μεταξύ τους για την επίτευξη ενός κοινού στόχου. Η προστασία των κοινών δομών δεδομένων είναι, λοιπόν, αρμοδιότητα των νημάτων.

Η έλλειψη προστασίας και η χρήση ενός κοινού χώρου διευθύνσεων για όλα τα νήματα μίας διεργασίας σημαίνει ότι το κόστος μεταγωγής των συμφραζομένων των νημάτων της ίδιας διεργασίας είναι πολύ χαμηλό, αφού το μόνο που απαιτείται είναι η τροποποίηση των καταχωρητών του επεξεργαστή. Δεν απαιτείται αλλαγή του χώρου διευθύνσεων, του πίνακα σελίδων ή/και τμημάτων, ακύρωση των περιεχομένων της κρυφής μνήμης ή ανάγνωση και εγγραφή δεδομένων στο δίσκο, αφου ο χώρος εικονικών διευθύνσεων της διεργασίας δεν αλλάζει. Η λειτουργία των νημάτων κάτω από το ίδιο καθεστώς ασφαλείας απαιτεί προσοχή στον

προγραμματισμό, αλλά επιτρέπει τον (κατα)μερισμό της κατάστασης ανάμεσα στα νήματα και την άμεση επικοινωνία τους μέσω της κοινής μνήμης.

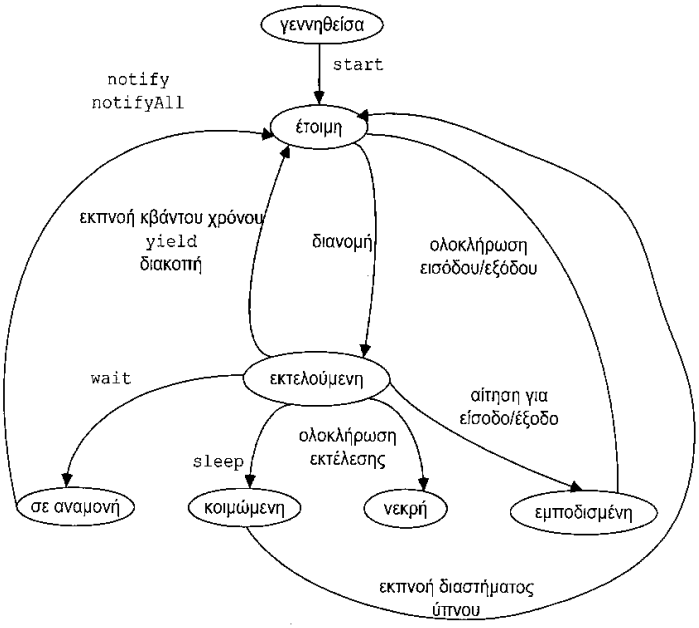
Στα συγκεντρωτικά συστήματα τα νήματα χρησιμοποιούνται επειδή διευκολύνουν τη συγγραφή εφαρμογών που εκτελούν παράλληλα πολλά καθήκοντα. Για παράδειγμα οι κλήσεις εισόδου και εξόδου μπορεί να προκαλέσουν εμποδισμό της διεργασίας που τις καλεί. Αν η εφαρμογή πρέπει να επικοινωνεί με το χρήστη και ταυτόχρονα να επεξεργάζεται κάποια δεδομένα, μπορούμε να χρησιμοποιήσουμε δυο νήματα ελέγχου, ένα για την επικοινωνία με το χρήστη και ένα για την επεξεργασία, προκειμένου να έχουμε ψευδοταυτόχρονη εκτέλεση και των δύο εργασιών.

Μια άλλη χρησιμότητα των νημάτων είναι η εύκολη αξιοποίηση συστημάτων πολυεπεξεργαστών με κοινή μνήμη σε εφαρμογές που επιδεικνύουν εγγενή παραλληλισμό. Ο παραλληλισμός μπορεί να εκφραστεί με πολλά νήματα ελέγχου, τα οποία θα εκτελούνται είτε σε έναν είτε σε πολλούς επεξεργαστές χωρίς καμία αλλαγή στην εφαρμογή. Αν χρησιμοποιούσαμε πολλές διεργασίες για τον ίδιο σκοπό, η ανάγκη επικοινωνίας τους μέσω του πυρήνα για ανταλλαγή δεδομένων θα οδηγούσε σε χαμηλή επίδοση λόγω της συχνής μεταγωγής συμφραζομένων. Αντίθετα τα νήματα επικοινωνούν χωρίς παρέμβαση του πυρήνα, χρησιμοποιώντας κοινές δομές δεδομένων στη μνήμη, με αποτέλεσμα την αύξηση της επίδοσης.

Η Java παρέχει πλήρη υποστήριξη σε επίπεδο γλώσσας προγραμματισμού για τα νήματα. Οι καταστάσεις ενός νήματος στην Java είναι παρόμοιες με αυτές μιας διεργασίας. Όταν ένα νήμα δημιουργηθεί βρίσκεται στη *γεννηθείσα* (*born*) *κατάσταση*. Το νήμα παραμένει στην κατάσταση αυτή μέχρις ότου ένα άλλο νήμα καλέσει τη μέθοδο `start` του πρώτου, η οποία αλλάζει την κατάστασή του σε *έτοιμη (ready)* ή *εκτελέσιμη (runnable)*. Το νήμα που έχει τη μεγαλύτερη προτεραιότητα μπαίνει στην *εκτελούμενη (running) κατάσταση*, δηλαδή αρχίζει την εκτέλεσή του, όταν το σύστημα το διανείμει σε έναν επεξεργαστή. Τέλος ένα νήμα μπαίνει στη *νεκρή (dead) κατάσταση*, όταν, για οποιονδήποτε λόγο, τελειώσει η εκτέλεσή του. Ένα νεκρό νήμα διαγράφεται από το σύστημα με τη διαδικασία της συλλογής απορριμμάτων(garbage collection).

Κατά τη διάρκεια της εκτέλεσής του ένα νήμα μπορεί να μπει στην *εμποδισμένη (blocked) κατάσταση*, αν χρειαστεί να περιμένει για είσοδο/έξοδο ή για κάποιο άλλο γεγονός. Όταν ολοκληρωθεί η είσοδος/έξοδος ή όταν συμβεί το γεγονός, η κατάσταση του νήματος μεταβάλλεται από εμποδισμένη σε έτοιμη. Ένα εκτελούμενο νήμα μπορεί να μπει οικειοθελώς στην *κοιμωμένη (sleeping) κατάσταση*, αν αποφασίσει να αναστείλει την εκτέλεσή του για κάποιο χρονικό διάστημα. Όταν περάσει το χρονικό διάστημα αυτό ή όταν κάποιο άλλο νήμα διακόψει τον ύπνο του, το αποκοιμισμένο νήμα γίνεται έτοιμο για εκτέλεση από το σύστημα.

Τέλος ένα νήμα μπορεί να μπει οικειοθελώς σε *κατάσταση αναμονής (waiting)*, αν επιθυμεί να περιμένει για την ικανοποίηση κάποιας συνθήκης σε ένα αντικείμενο, πιθανόν για περιορισμένο χρονικό διάστημα. Στην περίπτωση αυτήν η κατάσταση του νήματος αλλάζει σε έτοιμη είτε όταν κάποιο άλλο νήμα *γνωστοποιήσει (notify)* στο πρώτο (ή σε όλα τα νήματα που περιμένουν για κάποια συνθήκη στο ίδιο αντικείμενο) ότι η συνθήκη ικανοποιήθηκε, είτε όταν περάσει το χρονικό διάστημα αναμονής. Ο κύκλος ζωής ενός νήματος φαίνεται στο Σχήμα 8.



Σχήμα 8: Κατάσταση νημάτων.

Στην Java τα νήματα υποστηρίζονται μέσω της κλάσης `Thread` του πακέτου *java.lang*. Δύο κατασκευαστές της κλάσης αυτής είναι:

```
public Thread (String thread_name)
```

κατασκευάζει ένα νέο αντικείμενο τύπου `Thread` (ένα νήμα) το οποίο ονομάζεται `thread_name`. Τα ονόματα των νημάτων παρέχονται μόνο για ευκολία του προγραμματιστή, δηλαδή δε χρησιμοποιούνται από το σύστημα υποστήριξης της εκτέλεσης, αλλά κάθε νήμα πρέπει να έχει ένα όνομα. Έτσι ο κατασκευαστής:

```
public Thread ()
```

κατασκευάζει ένα αντικείμενο τύπου `Thread` που έχει όνομα "`Thread-#`" όπου `#` είναι ο αύξων αριθμός του νήματος αυτού (π.χ. `Thread-1`, `Thread-2` κ.ο.κ.). Έτσι στην Java δεν έχουμε ποτέ ανώνυμα νήματα.

Μερικές από τις μεθόδους της κλάσης `Thread` είναι:

```
public void start() throws IllegalThreadStateException
```

ξεκινά ένα νήμα, δηλαδή η εικονική μηχανή της Java καλεί τη μέθοδο `run` του νήματος. Μετά την κλήση και τα δύο νήματα (το νέο νήμα και το νήμα που το ξεκίνησε) εκτελούνται παράλληλα. Η μέθοδος αυτή παράγει την εξαίρεση `IllegalThreadStateException`, αν το επιθυμητό νήμα έχει ήδη ξεκινήσει.

```
public void start()
```

είναι η μέθοδος μέσα στην οποία υπάρχουν οι εντολές εκτέλεσης του νήματος. Ένα νήμα πρέπει είτε να ακυρώσει τη μέθοδο αυτήν είτε να υλοποιήσει τη μέθοδο `run()` της διεπαφής `Runnable` (την οποία υλοποιεί η κλάση `Thread`).

```
public final void setName (String name)
```

αλλάζει το όνομα του νήματος σε `name`. Παράγει μια εξαίρεση τύπου `SecurityException`, αν το τρέχον νήμα δεν έχει δικαίωμα να τροποποιήσει το όνομα του συγκεκριμένου νήματος.

```
public final String getName ()
```

επιστρέφει το όνομα του νήματος.

```
public static Thread currentThread ()
```

επιστρέφει μια αναφορά προς το τρέχον (εκτελούμενο) νήμα.

```
public static void sleep (long millis)
                                    throws InterruptedException
```

αποκοιμίζει το νήμα αυτό για διάστημα `millis` σε msec. Παράγει μία εξαίρεση τύπου `InterruptedException`, αν κάποιο άλλο νήμα διακόψει το νήμα αυτό.

```
public void interrupt ()
```

στέλνει στο νήμα αυτό ένα σήμα διακοπής. Αν το νήμα είναι σε αποκοιμισμένη κατάσταση (έχει καλέσει τη μέθοδο `sleep`) ή σε κατάσταση αναμονής (έχει καλέσει τη μέθοδο `wait`), παράγεται μία εξαίρεση τύπου `InterruptedException`. Σε κάθε άλλη περίπτωση το νήμα δε διακόπτεται, οπότε πρέπει το ίδιο να εξετάζει περιοδικά αν του έχει σταλεί κάποιο σήμα διακοπής. Αν το νήμα δεν κάνει κάτι τέτοιο, τα σήματα διακοπής που του στέλνονται αγνοούνται. Η μέθοδος αυτή παράγει μια εξαίρεση τύπου `SecurityException`, αν το τρέχον νήμα δεν έχει το δικαίωμα να διακόψει το νήμα αυτό.

```
public boolean isInterrupted ()
```

επιστρέφει την τιμή `true`, αν το τρέχον νήμα έχει διακοπεί, ειδάλλως επιστρέφει την τιμή `false`. Με τη μέθοδο αυτήν ένα νήμα μπορεί να εξετάζει περιοδικά κατά πόσο του έχει σταλεί ένα σήμα διακοπής.

Υπάρχουν δύο τρόποι για την δημιουργία ενός νήματος(Thread).

Ο πρώτος είναι παρέχοντας ένα τύπου `Runnable` αντικείμενο. Το interface `Runnable` διαθέτει τη μέθοδο `run` η οποία έχει ως σκοπό να περιέχει τον κώδικα που θα εκτελεστεί στο thread. Το τύπου `Runnable` αντικείμενο περνιέται σαν όρισμα στον κατασκευαστή του Thread όπως στο παράδειγμα `HelloRunnable` παρακάτω:

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

Ο δεύτερος τρόπος είναι επεκτείνοντας(κληρονομώντας) την κλάση `Thread`. Η κλάση `Thread` υλοποιεί από μόνη της το interface `Runnable`, χωρίς να υλοποιεί την μέθοδο `run`. Μια εφαρμογή κληρονομεί την κλάση `Thread`, παρέχοντας την δική της υλοποίηση για την μέθοδο `run`, όπως στο παράδειγμα `HelloThread` παρακάτω:

```
public class HelloThread extends Thread {

       public void run() {
              System.out.println("Hello from a thread!");
       }

       public static void main(String args[]) {
              (new HelloThread()).start();
       }

}
```

2.6. Η κλάση InetAddress

Αυτή είναι η κλάση που χρησιμοποιείται για την διαχείριση των διευθύνσεων IP, οι οποίες είναι μοναδικές ακολουθίες 32(διευθύνσεις IPv4) ή 128 (διευθύνσεις IPv6) *bits* που χρησιμοποιούμε για να απευθυνόμαστε στους ξένιους υπολογιστές στο Διαδίκτυο. Η `InetAddress` έχει δύο υποκλάσεις, τις `Inet4Address` και `Inet6Address`, οι οποίες παριστάνουν ειδικά τις διευθύνσεις IPv4 και IPv6, αντίστοιχα. Κάθε αντικείμενο τύπου `InetAddress` αποτελείται από μία διεύθυνση IP και πιθανώς από το αντίστοιχο όνομα ξένιου υπολογιστή (host name). Η κλάση `InetAddress` παρέχει μεθόδους για την αντιστοίχιση ονομάτων ξένιων υπολογιστών στις IP διευθύνσεις τους και αντίστροφα, μέσω κάποιας υπηρεσίας ονομασίας, όπως το DNS.

Μερικές από τις μεθόδους της κλάσης `InetAddress` είναι οι:

```
public static InetAddress getByAddress (byte[] addr)
                                    throws UnknownHostException
```

όπου addr είναι ένας πίνακας τεσσάρων (IPv4) ή δεκαέξι (IPv6) bits συλλαβών, ο οποίος παριστάνει μία διεύθυνση IP σε διάταξη bit-συλλαβών δικτύου (network byte order), δηλαδή η πιο σημαντική bit-συλλαβή της διεύθυνσης IP βρίσκεται στη θέση 0 του πίνακα addr. Η μέθοδος αυτή επιστρέφει ένα αντικείμενο τύπου `InetAddress` με βάση τη διεύθυνση IP που δίνεται. Συγκεκριμένα, αν η παράμετρος *addr* προσδιορίζει μια διεύθυνση IPv4, τότε επιστρέφεται ένα αντικείμενο τύπου `Inet4Address`, διαφορετικά, σε περίπτωση διευθύνσεων IPv6, επιστρέφεται αντικείμενο τύπου `Inet6Address`. Αν η διεύθυνση που δίνεται δεν έχει το σωστό μέγεθος (4 ή 16 bit-συλλαβές), τότε παράγεται μια εξαίρεση τύπου `UnknownHostException`. Η μέθοδος αυτή δεν επιλύει (resolve) τη διεύθυνση *addr*, δε βρίσκει δηλαδή το αντίστοιχο όνομα του ξένιου υπολογιστή.

```
public static InetAddress getByAddress (String host, byte[] addr)
                                    throws UnknownHostException
```

όπου η συμβολοσειρά *host* μπορεί να είναι είτε το όνομα ενός υπολογιστή, Π.χ. "nefeli.teicrete.gr", είτε η IP διεύθυνσή του υπό μορφή κειμένου, Π.χ. "205.150.23.58.". Για την παράμετρο *addr* ισχύουν τα παραπάνω σχόλια. Η μέθοδος αυτή ελέγχει μόνο αν το μήκος της IP διεύθυνσης που δίνεται είναι νόμιμο και, αν δεν είναι, παράγει μία εξαίρεση τύπου `UnknownHostException`. Δεν εκτελείται κανένας άλλος έλεγχος επί των παραμέτρων. Επιστρέφει ένα αντικείμενο τύπου `InetAddress` με βάση τις παραμέτρους της μεθόδου.

```
public static InetAddress getByName (String host)
                                    throws UnknownHostException
```

καθορίζει την IP διεύθυνση ενός ξένιου υπολογιστή με βάση το όνομά του. Για τη συμβολοσειρά *host* ισχύουν τα παραπάνω σχόλια, με τη διαφορά ότι σε περίπτωση που αυτή περιέχει μια διεύθυνση IP σε μορφή κειμένου, ελέγχεται απλά το μορφότυπο της διεύθυνσης (χωρίς να εντοπίζεται το όνομα της συγκεκριμένης μηχανής). Σε περίπτωση που η παράμετρος *host* περιέχει το όνομα ενός ξένιου υπολογιστή, παράγεται μια εξαίρεση τύπου `UnknownHostException` σε περίπτωση που δεν κατέστη δυνατό να εντοπιστεί η διεύθυνση IP του.

```
public InetAddress getLocalHost () throws UnknownHostException
```

επιστρέφει τη διεύθυνση IP του τοπικού ξένιου υπολογιστή ή παράγει μια εξαίρεση τύπου `UnknownHostException,` αν αυτή δεν μπορεί να εντοπιστεί.

```
public String getHostAddress ()
```

επιστρέφει τη διεύθυνση IP σε μορφή κειμένου.

```
public String getHostName ()
```

επιστρέφει το όνομα του ξένιου υπολογιστή γι' αυτήν τη διεύθυνση IP. Αν το συγκεκριμένο αντικείμενο της κλάσης `InetAddress` έχει δημιουργηθεί με βάση ένα όνομα ξένιου υπολογιστή (π.χ. με χρήση της μεθόδου `getByName`), τότε η μέθοδος `getHostName` επιστρέφει το όνομα αυτό. Σε κάθε άλλη περίπτωση, αν δηλαδή το αντικείμενο έχει δημιουργηθεί με βάση μία διεύθυνση IP, το όνομα εντοπίζεται μέσω κάποιας υπηρεσίας ονομασίας και επιστρέφεται ή, αν αυτό δεν είναι δυνατόν, επιστρέφεται η ίδια η διεύθυνση IP σε μορφή κειμένου.

```
public String getHostName ()
```

επιστρέφει μια συμβολοσειρά της μορφής *όνομα_ξένιου_υπολογιστή/διεύθυνση IP*. Σε περίπτωση που η διεύθυνση IP δεν έχει επιλυθεί σε όνομα το μέρος του ονόματος είναι η κενή συμβολοσειρά.

2.7. Δομές Map (κλάση Hashtable)

Πρόκειται για δομές που χρησιμοποιούνται στα πλαίσια της αποθήκευσης και διαχείρισης μεγάλης και σύνθετης πληροφορίας(συλλογές από δεδομένα, κλπ). Η κλάση Hashtable υλοποιεί Map δομή η οποία αποθηκεύει δεδομένα σε ζευγάρια key/value. Δίνοντας το πεδίο key παίρνεις εύκολα και γρήγορα την τιμή του πεδίου value. Ένα αντικείμενο hashtable πρέπει να έχει μοναδικές τιμές για κάθε πεδίο key(αναγνωριστικό γνώρισμα) ενώ το πεδίο value μπορεί να παίρνει τιμές που επαναλαμβάνονται. Ένας από τους κατασκευαστές της κλάσης αυτής είναι:

```
public Hashtable ()
```

κατασκευάζει ένα νέο αντικείμενο τύπου `Hashtable` (ένα νήμα) το οποίο αρχικά είναι άδειο.

Μερικές από τις μεθόδους της κλάσης `Hashtable` είναι:

```
public Object put(Object key, Object value)
```

συνδέει(maps) το συγκεκριμένο `key` με το συγκεκριμένο `value` σε αυτό το `hashtable`. Το `key` και το `value` δεν μπορούν να πάρουν *null* τιμές. Το `value` μπορεί να ανακτηθεί μέσω της μεθόδου **`get`** με ένα κλειδί το οποίο είναι ίδιο με αυτό που έχει αρχικά αποθηκευτεί.

```
public Object get(Object key)
```

επιστρέφει το `value` με το οποίο είναι συνδεδεμένο(mapped) το συγκεκριμένο `key` ή *null* εάν δεν υπάρχει καμία σύνδεση(mapping) με αυτό το κλειδί. Υπάρχει το πολύ μία μόνο τέτοια διασύνδεση(mapping).

```
public Object remove(Object key)
```

διαγράφει το `key` και το αντίστοιχο `value` με το οποίο είναι συνδεδεμένο(mapped) σε αυτό το `Hashtable`. Η μέθοδος αυτή δεν κάνει τίποτα εάν το `key` δεν υπάρχει σε αυτό το `Hashtable`. Επιστρέφει το `value` με το οποίο το `key` είχε συνδεθεί σε αυτό το `Hashtable` ή *null* εάν για αυτό το κλειδί δεν υπάρχει διασύνδεση(mapping).

```
public boolean containsKey(Object key)
```

επιστρέφει την τιμή `true`, εάν το `key` υπάρχει σε αυτό το `Hashtable`, ειδάλλως επιστρέφει την τιμή `false`.

```
public boolean containsValue(Object value)
```

επιστρέφει την τιμή `true`, εάν κάποιο κλειδί είναι συνδεδεμένο(mapped) με το συγκεκριμένο `value` σε αυτό το `Hashtable`, ειδάλλως επιστρέφει την τιμή `false`.

Η ανάκτηση όλων των ζευγών key/value(και όχι μεμονωμένα ενός value μέσω ενός key με χρήση της μεθόδου get) μπορεί να γίνει κάνοντας χρήση του αντικείμενου `Iterator` όπως στο παράδειγμα `DemoHashtable` παρακάτω:

```
public class DemoHashtable {

	public static void main(String args[]) {

		Hashtable courses = new Hashtable();
		courses.put("CS-1000", "Introduction to Informatics");
		courses.put("CS-2001", "Programming");
		courses.put("CS-3004", "Databases");
		courses.put("CS-4002", "Object Oriented Programming");
		courses.put("CS-6001", "Web Programming");


		Iterator coursesIterator = courses.keySet().iterator();
		while (coursesIterator.hasNext()) {
			//Τρέχων τιμή του key (κωδικός μαθήματος).
			String courseKey = (String) coursesIterator.next();
			//Τρέχων τιμή του value (όνομα μαθήματος).
			String courseName = (String) courses.get(courseKey);

			System.out.println(courseKey+"-"+courseName);
		}
	}

}
```

3. Πρωτόκολλο επικοινωνίας UDP

3.1. Εισαγωγή

Το πρωτόκολλο User Datagram Protocol (UDP) η εναλλακτικά Universal Datagram Protocol είναι από τα βασικά πρωτόκολλα που χρησιμοποιούνται σήμερα σε πληθώρα εφαρμογών στο διαδίκτυο. Πρόκειται για πρωτόκολλο μεταφοράς χωρίς συνδέσεις που σημαίνει ότι δεν εγγυάται την παράδοση των πακέτων η την αποστολή αυτών σε σειρά. Στο UDP τα bytes της πληροφορίας αντί να γράφονται η να διαβάζονται σε μια σειρά σε κανάλια μεταφοράς(streams), ομαδοποιούνται σε πακέτα τα οποία και στέλνονται στον παραλήπτη μέσω του δικτύου.

Τα πακέτα μπορούν να ταξιδεύουν από διαφορετικά μονοπάτια, τα οποία επιλέγονται από τους διάφορους δρομολογητές δικτύου ανάλογα με παράγοντες όπως η συμφόρηση του δικτύου, προτεραιότητα των δρομολογίων, καθώς και το κόστος μεταφοράς.

Αυτό σημαίνει ότι ένα πακέτο μπορεί να φτάσει εκτός ακολουθίας, εάν για παράδειγμα ακολουθήσει μια γρηγορότερη διαδρομή από ότι ένα άλλο πακέτο (ή εάν το άλλο πακέτο συναντήσει κάποιας μορφής καθυστέρηση). Μάλιστα εάν σε μια συγκεκριμένη διαδρομή υπάρχει μεγάλη συμφόρηση, το πακέτο ενδέχεται να απορριφθεί εξ ολοκλήρου. Όμως, εάν ένα πακέτο φτάσει, θα είναι άθικτο γιατί τα πακέτα που είναι κατεστραμμένα(corrupted) ή εν μέρει ακέραια απορρίπτονται.

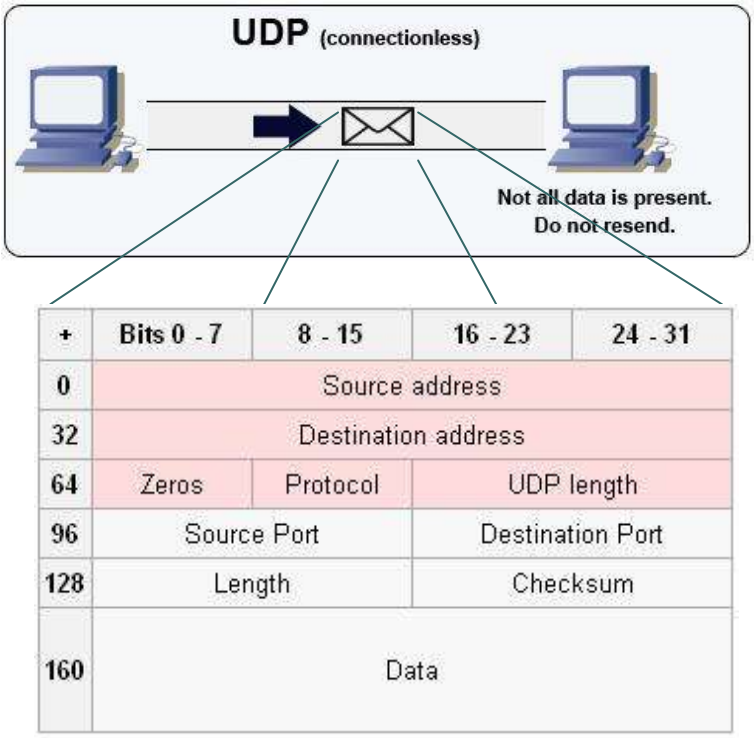
Λόγω της δυνητικής για απώλεια των πακέτων δεδομένων, μπορεί να φαίνεται περίεργο το γεγονός ότι κάποιος θα μπορούσε ακόμη και να εξετάσει την χρήση ενός τόσο αναξιόπιστου, φαινομενικά συστήματος. Στην πραγματικότητα, υπάρχουν πολλά πλεονεκτήματα από τη χρήση UDP που μπορεί να μην είναι εμφανής με την πρώτη ματιά.

- Η UDP επικοινωνία μπορεί να είναι πιο αποτελεσματική από την εγγυημένη παράδοση με κανάλια δεδομένων(data streams). Εάν η όγκος των δεδομένων είναι μικρός και τα δεδομένα αποστέλλονται συχνά, η χρήση της θα μπορούσε να αποκτήσει νόημα ώστε να αποφευχθεί η επιβάρυνση από τις διαδικασίες της εγγυημένης παράδοσης.
- Σε αντίθεση με τα κανάλια δεδομένων(data streams) στα οποία απαιτείται σύνδεση, το UDP μπορεί να αποδειχτεί χρησιμότερο λόγω μικρότερου κόστους. Εάν η όγκος των δεδομένων είναι μικρός και τα δεδομένα αποστέλλονται σπάνια, η διαδικασία δημιουργίας σύνδεσης ίσως να μην αξίζει να πραγματοποιηθεί. Το UDP μπορεί να είναι καλύτερο σε αυτή την περίπτωση, ιδιαίτερα αν τα δεδομένα αποστέλλονται από ένα μεγάλο αριθμό μονάδων σε ένα κεντρικό, οπότε το συνολικό άθροισμα όλων αυτών των συνδέσεων μπορεί να προκαλέσει σημαντική υπερφόρτωση.
- Εφαρμογές Real-time(πραγματικού χρόνου) που απαιτούν καλές επιδόσεις, είναι υποψήφιες για χρήση UDP, καθώς υπάρχουν λιγότερες καθυστερήσεις όπως αυτές που οφείλονται στον έλεγχο σφαλμάτων και ροής του ελέγχου του πρωτοκόλλου TCP. Τα UDP πακέτα μπορούν να χρησιμοποιηθούν για τον κορεσμό του διαθέσιμου εύρους ζώνης του δικτύου για την παροχή μεγάλων ποσοτήτων δεδομένων (όπως streaming video / audio, ή τηλεμετρίας

δεδομένων για πολλούς παίκτες σε ένα παιχνίδι δικτύου, κλπ). Επιπλέον, εάν κάποια δεδομένα χάνονται, μπορεί να αντικατασταθούν από την επόμενη σειρά πακέτων με ανανεωμένη πληροφορία, εξαλείφοντας την ανάγκη να σταλούν εκ νέου τα παλιά δεδομένα που πλέον είναι μη χρήσιμα.

- Οι υποδοχές στο UDP μπορούν να λαμβάνουν δεδομένα από περισσότερα του ενός μηχανήματα. Εάν η επικοινωνία είναι μεταξύ πολλών μηχανημάτων, τότε το UDP ίσως να είναι η πιο εύκολη λύση σε σχέση με άλλους μηχανισμούς, όπως το TCP.

Η δομή των μηνυμάτων (γνωστών και ως δεδομενογραφήματα-datagrams) που αποστέλλονται στο πρωτόκολλο UDP από μία υπολογιστική μονάδα σε μια άλλη μέσα σε ένα δίκτυο υπολογιστών αποτελείται από την επικεφαλίδα(source port, destination port, UDP length, UDP checksum) και την ωφέλιμη πληροφορία όπως στο Σχήμα 9.



Σχήμα 9: Δομή UDP πακέτου

Στην Java ένα δεδομενογράφημα, υλοποιείται από την κλάση `DatagramPacket` του πακέτου java.net. Δύο κατασκευαστές της κλάσης αυτής είναι:

```
public DatagramPacket (byte[] buff, int length)
```

δημιουργεί ένα αντικείμενο τύπου `DatagramPacket` για τη λήψη πακέτων μεγέθους `length`, Η τιμή της παραμέτρου `length` πρέπει να είναι μικρότερη ή ίση με το μέγεθος του πίνακα `buff`.

```
public DatagramPacket(byte[] buff, int length,
InetAddress address, int port)
```

δημιουργεί ένα αντικείμενο τύπου `DatagramPacket` για την αποστολή πακέτων μεγέθους `length` στη θύρα `port` του ξένιου υπολογιστή `host`. Για την τιμή της παραμέτρου `length` ισχύουν τα παραπάνω σχόλια

Μερικές από τις μεθόδους της κλάσης `DatagramPacket` είναι:

```
public InetAddress getAddress ()
```

επιστρέφει τη διεύθυνση IP της μηχανή; στην οποία στέλνεται ή από την οποία προέρχεται το συγκεκριμένο δεδομενογράφημα.

```
public void setAddress (InetAddress addr)
```

θέτει τη διεύθυνση IP της μηχανής στην οποία στέλνεται το συγκεκριμένο δεδομενογράφημα.

```
public int getPort ()
```

επιστρέφει τον αριθμό τοπικής θύρας της μηχανής στην οποία στέλνεται ή από την οποία προέρχεται το συγκεκριμένο δεδομενογράφημα.

```
public void setPort (int port)
```

θέτει τον αριθμό της τοπικής θύρας της απομακρυσμένης μηχανής στην οποία στέλνεται το συγκεκριμένο δεδομενογράφημα.

```
public byte[] getData ()
```

επιστρέφει τα δεδομένα του συγκεκριμένου δεδομενογραφήματος

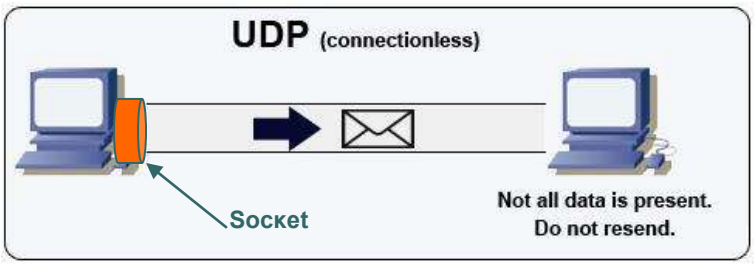
```
public void setData (byte [] buff)
```

θέτει τα δεδομένα του συγκεκριμένου δεδομενογραφήματος.

```
public int getLength()
```

επιστρέφει το μέγεθος length των δεδομένων που είναι να σταλούν η των δεδομένων που ελήφθησαν.

Η επικοινωνία μεταξύ διεργασιών που πιθανώς βρίσκονται σε ξεχωριστές υπολογιστικές μονάδες, σύμφωνα με το πρωτόκολλο UDP, γίνεται μέσω των υποδοχών. Οι υποδοχές (sockets) όπως έχει ήδη αναφερθεί παρέχουν σημειακή, αμφίδρομη επικοινωνία μεταξύ διεργασιών, συνήθως μέσω δικτύου. Αποτελούν μια πολύ ευέλικτη και θεμελιώδη μέθοδο δικτυακής επικοινωνίας μεταξύ διεργασιών ή συστημάτων. Στο UDP η υποδοχή(datagram socket) δεν απαιτεί την δημιουργία σύνδεσης με τον αποδέκτη για την αποστολή των πακέτων αφού κάθε μήνυμα μεταφέρει μαζί του την διεύθυνση προορισμού.



Σχήμα 10: Υποδοχές UDP

Για την επικοινωνία, κάθε υπολογιστική μονάδα πρέπει να διαθέτει μια υποδοχή(datagram socket) συνδεδεμένη σε μια θύρα(port) της μονάδας αυτής. Για την αποστολή των μηνυμάτων(datagram packets) ο αποστολέας συμπληρώνει στην επικεφαλίδα τα πεδία που χρειάζονται και το πακέτο μαζί με την ωφέλιμη πληροφορία προωθείται μέσω των υποδοχών (sockets) στο IP επιπέδου διαδικτύου και εν συνεχεία στον αποδέκτη.

Στην JAVA η UDP υποδοχή υλοποιείται από την κλάση `DatagramSocket` του πακέτου java.net. Δύο κατασκευαστές της κλάσης αυτής είναι:

```
public DatagramSocket () throws SocketException
```

κατασκευάζει μία υποδοχή δεδομενογραφημάτων και τη συνδέει σε κάποια (οποιαδήποτε) θύρα στην τοπική μηχανή. Αν η υποδοχή δεν μπορεί να ανοιχτεί, παράγεται μία εξαίρεση τύπου `SocketException`.

```
public DatagramSocket (int port) throws SocketException
```

κατασκευάζει μία υποδοχή δεδομενογραφημάτων, και τη συνδέει στη θύρα `port` της τοπικής μηχανής. Αν η υποδοχή δεν μπορεί να ανοιχτεί ή δεν μπορεί να δεσμευτεί

στη συγκεκριμένη θύρα που δίνεται, παράγεται μία εξαίρεση τύπου `SocketException`.

Μερικές από τις μεθόδους της κλάσης `DatagramSocket` είναι:

```
public void send (DatagramPacket packet) throws IOException
```

στέλνει ένα δεδομενογράφημα μέσω της συγκεκριμένης υποδοχής. Όπως είδαμε, το δεδομενογράφημα περιέχει εκτός των δεδομένων, το μήκος τους, καθώς και τη διεύθυνση και τον αριθμό της θύρας του απομακρυσμένου υπολογιστή στον οποίον εκτελείται ο εξυπηρετητής, στον οποίον απευθύνεται.

```
public void receive (DatagramPacket packet) throws IOException
```

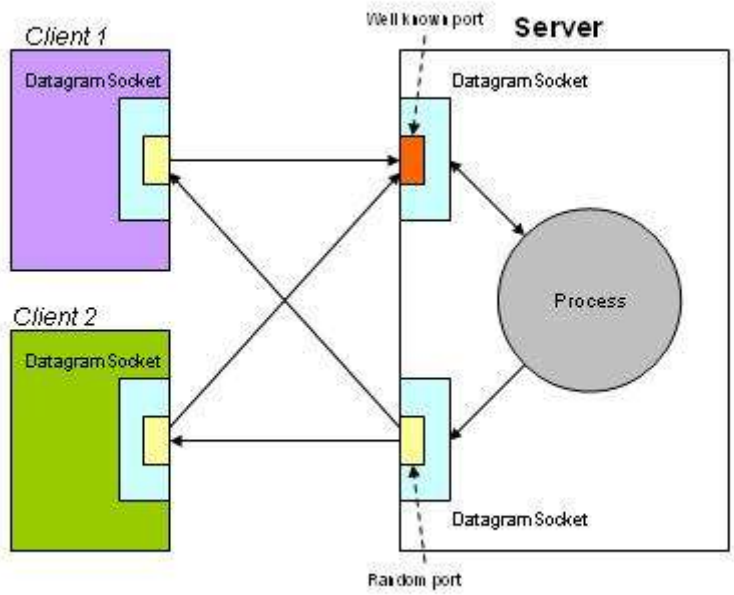
παραλαμβάνει ένα δεδομενογράφημα μέσω της συγκεκριμένης υποδοχής. Όταν η μέθοδος επιστρέφει, η παράμετρος `packet` περιέχει τόσο τα δεδομένα και το μήκος τους, όσο και τη διεύθυνση και τον αριθμό της θύρας της μηχανής του αποστολέα. Η μέθοδος εμποδίζεται, μέχρις ότου αφιχθεί κάποιο πακέτο. Το πεδίο `length` του αντικειμένου `packet` περιέχει το μήκος των δεδομένων που παρελήφθησαν. Αν τα δεδομένα που εστάλησαν είχαν μεγαλύτερο μήκος από τον ενταμιευτή του δεδομενογραφήματος `packet`, τότε περικόπτονται.

```
public void close ()
```

κλείνει τη συγκεκριμένη υποδοχή. Κάθε νήμα που είναι εμποδισμένο πάνω στην υποδοχή αυτή παράγει μια εξαίρεση τύπου `SocketException`.

3.2. Μηχανισμός Client - Server

Υλοποιώντας σε Java μία Client-Server εφαρμογή στο UDP(Σχήμα 11), η διεργασία στον εξυπηρετητή (server process) θα πρέπει αρχικά να αφουγκράζεται για εισερχόμενα πακέτα σε μία συγκεκριμένη πόρτα «γνωστή» στους πελάτες(clients). Έτσι δημιουργούμε ένα `DatagramSocket` με συγκεκριμένο αριθμό πόρτας(well known port). Στον πελάτη δημιουργούμε αντίστοιχα ένα `DatagramSocket` και στέλνουμε στον εξυπηρετητή UDP πακέτα ορίζοντας ως πόρτα προορισμού τον συγκεκριμένο αριθμό πόρτας στον οποίο «ακούει» ο εξυπηρετητής. Ωστόσο αυτή την φορά δεν χρειάζεται να ορίσουμε στον πελάτη συγκεκριμένο αριθμό τοπικής πόρτας. Αφήνουμε το σύστημα να δεσμεύσει την πρώτη ελεύθερη port που υπάρχει διαθέσιμη και μέσω των μηνυμάτων που θα σταλθούν στον εξυπηρετητή αυτή θα γίνει γνωστή σε αυτόν, ώστε στην συνέχεια αν χρειαστεί να μπορεί ο πελάτης να λάβει απαντήσεις από αυτόν. Στον εξυπηρετητή για κάθε εισερχόμενο πακέτο η διεργασία μας μπορεί να προσδιορίσει ποιος ήταν ο αποστολέας (source address, port) και να εκτελέσει τις κατάλληλες ενέργειες σύμφωνα με τον σκοπό για τον οποίο δημιουργήθηκε. Αν ο εξυπηρετητής χρειάζεται να στείλει απαντήσεις στον αποστολέα, το κάνει δημιουργώντας μία νέα υποδοχή(datagram socket), στέλνοντας UDP πακέτα από εκεί, στην γνωστή πλέον διεύθυνση του πελάτη.



Σχήμα 11: Μηχανισμός UDP Client-Server

Παρακάτω παρατίθεται κώδικας σε Java για μονόδρομη επικοινωνία όπου ο πελάτης(client) στέλνει μηνύματα στον εξυπηρετητή(server) και αυτός τα εκτυπώνει στην έξοδο. Σε περίπτωση που θέλουμε η επικοινωνία να είναι αμφίδρομη ακολουθούμε αντίστοιχες διαδικασίες(receive, send) και στις δύο πλευρές(client-server), ωστόσο θα χρειαστεί να υλοποιηθούν οι διαδικασίες αυτές σε νήματα (`Thread`) για να μπορούν να εκτελεστούν (ψευδο)παράλληλα σε κάθε πλευρά.

Αρχικά ο κώδικας του εξυπηρετητή:

```
public class Server {

	private int serverPort = 10000;

	private DatagramSocket socket;

	public Server() {
		try {
			socket = new DatagramSocket(serverPort);
		} catch (SocketException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public void receiveData() {

		byte[] data = new byte[2000];

		System.out.println("Waiting...");

		while (true) {
			DatagramPacket packet = new DatagramPacket(data,
				data.length);
			try {
				socket.receive(packet);

				ByteArrayInputStream bais = new
					ByteArrayInputStream(packet.getData(),
					0, packet.getLength());
				BufferedReader reader = new BufferedReader(new
					InputStreamReader(bais));

				System.out.println(packet.getAddress().getHostA
					ddress() + " : " + reader.readLine());
			} catch (IOException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
	}

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

Ακολουθεί ο κώδικας του πελάτη:

```
public class Client {
	private static InetAddress serverAddress;

	private int serverPort = 10000;

	private DatagramSocket socket;

	public Client() {
		try {
			socket = new DatagramSocket();
		} catch (SocketException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public void sendData() {

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

		while (true) {
			try {
				String data = br.readLine();

				byte[] dataToSend = data.getBytes();

				DatagramPacket packet = new
					DatagramPacket(dataToSend,
					dataToSend.length, serverAddress,
					serverPort);

				socket.send(packet);
			} catch (IOException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
	}

	public static void main(String[] args) {
		if (args.length != 1) {
			System.out.println("Error: you forgot <server IP>");
			return;
		}

		String serverIP = args[0];

		try {
			serverAddress = InetAddress.getByName(serverIP);
		} catch (UnknownHostException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		Client client = new Client();
		client.sendData();
	}
}
```

3.3. Ανάπτυξη UDP εφαρμογής

Να υλοποιηθεί Client – Server μηχανισμός σύμφωνα με το πρωτόκολλο δικτύου UDP κατά τον οποίο ο client θα διαβάζει μηνύματα από το πληκτρολόγιο(ή textfield σε περίπτωση που τον υλοποιήσετε με διεπαφή) ,θα τα στέλνει στον server και αυτός θα τα προωθεί σε όλους τους συνδεμένους clients όπου και θα τα εκτυπώνουν. Τα μηνύματα που θα αποστέλλονται θα είναι τυποποιημένα της μορφής:

a) Login <username>
b) Message <message>
c) Logout

Ο server θα πράττει, αναλόγως τα προθέματα, ως εξής:

a) Θα αποθηκεύει το όνομα του user και την IP του σε μια λίστα με ενεργούς χρήστες.
β) Θα στέλνει το <message> σε όλους τους ενεργούς χρήστες.
c) Θα σβήνει το χρήστη από την λίστα των ενεργών χρηστών.

Προσέξτε τα παρακάτω

Να ορίσετε συγκεκριμένες πόρτες στις οποίες θα ακούει ο server και οι clients.

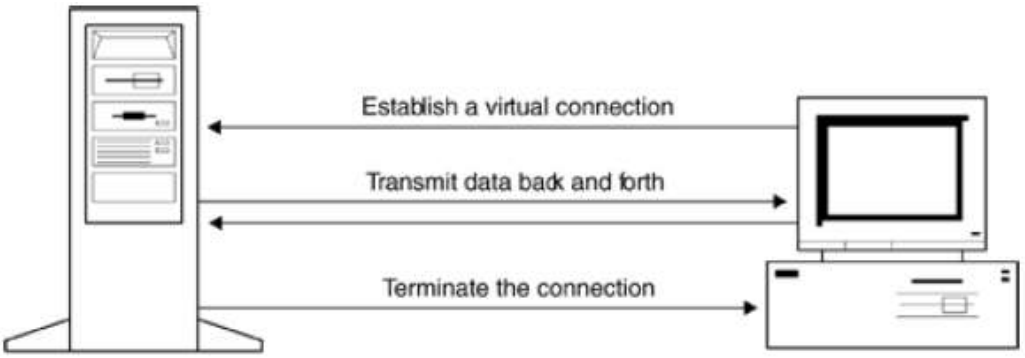
Η επικοινωνία θα είναι αμφίδρομη που σημαίνει ότι ο client και ο server θα έχουν μία μέθοδο για την αποστολή των μηνυμάτων και μία για την λήψη, και ότι μία από της δύο θα χρησιμοποιεί ένα νέο Thread.

4. Πρωτόκολλο επικοινωνίας TCP

4.1. Εισαγωγή

Το πρωτόκολλο TCP (Transmission Control Protocol - Πρωτόκολλο Ελέγχου Μεταφοράς) είναι ίσως το κυριότερο πρωτόκολλο που χρησιμοποιείται σήμερα στο διαδίκτυο. Κύριος στόχος του TCP είναι η αξιόπιστη αποστολή και λήψη δεδομένων έτσι ώστε ο παραλήπτης να λαμβάνει τα δεδομένα χωρίς λάθη και στη σωστή σειρά. Οι περισσότερες υπηρεσίες στο διαδίκτυο σήμερα βασίζονται στο TCP.

Το TCP παρέχει μια διασύνδεση στην επικοινωνία των δίκτυων, που είναι ριζικά διαφορετική από αυτή του User Datagram Protocol (UDP). Οι ιδιότητες του TCP το καθιστούν ιδιαίτερα ελκυστικό στους προγραμματιστές του δικτύου, δεδομένου ότι διευκολύνει την επικοινωνία σε ένα δίκτυο με την εξάλειψη πολλών από των εμποδίων του UDP, όπως παραγγελία των πακέτων και απώλεια αυτών. Ενώ το UDP αφορά την μετάδοση των πακέτων δεδομένων, το TCP, επικεντρώνεται στην δημιουργία μιας σύνδεσης δικτύου, μέσω της οποίας μια ροή από bytes είναι δυνατόν να αποστέλλεται και να λαμβάνεται. Τα πακέτα μπορούν να σταλούν μέσω ενός δικτύου χρησιμοποιώντας διάφορα μονοπάτια και μπορούν να καταλήξουν σε διαφορετικές χρονικές στιγμές. Αυτό παρέχει απόδοση και αντοχή, αφού η απώλεια ενός πακέτου δεν διαταράσσει κατ 'ανάγκη τη μετάδοση άλλων πακέτων. Ωστόσο, ένα τέτοιο σύστημα δημιουργεί επιπλέον εργασία για τους προγραμματιστές που θέλουν εγγυημένη παράδοση των δεδομένων. Το TCP εξαλείφει αυτή τη πρόσθετη εργασία, εξασφαλίζοντας την παράδοση και την ορθή σειρά, παρέχοντας μια αξιόπιστη ροή μεταφοράς πληροφορίας μεταξύ πελάτη και διακομιστή που υποστηρίζει αμφίδρομη επικοινωνία. Θεσπίζει μία "εικονική σύνδεση" μεταξύ των δύο μηχανημάτων, μέσω της οποίας οι ροές δεδομένων μπορούν να αποσταλούν(Σχήμα 12).



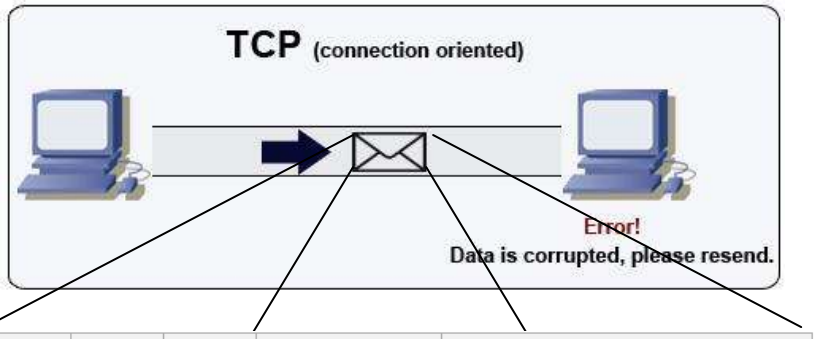
Σχήμα 12: Λειτουργία εικονικών συνδέσεων

Το TCP χρησιμοποιεί ένα χαμηλότερου επιπέδου πρωτόκολλο επικοινωνίας, το Internet Protocol (IP), για τον καθορισμό της σύνδεσης μεταξύ των τερματικών. Η σύνδεση αυτή παρέχει μια διεπαφή που επιτρέπει ροές από bytes να αποστέλλονται και να λαμβάνονται, και μετατρέπει με διαφάνεια τα δεδομένα σε IP δεδομενογραφήματα. Αν και πρόβλημα των datagrams, είναι ότι δεν υπάρχει εγγύηση ότι τα πακέτα θα φθάσουν στον προορισμό τους το TCP φροντίζει για αυτό το πρόβλημα παρέχοντας εγγυημένη παράδοση των bytes των δεδομένων. Φυσικά, είναι

πάντα πιθανό να υπάρχουν σφάλματα του δικτύου που θα εμποδίσουν την παράδοση, αλλά το TCP χειρίζεται τέτοια θέματα εφαρμογής όπως με το να ξαναστείλει πακέτα, και ειδοποιεί τον προγραμματιστή μόνο σε σοβαρές περιπτώσεις όπως εάν δεν υπάρχει καμία διαδρομή προς έναν υπολογιστή δικτύου, ή εάν η σύνδεση έχει χαθεί.

Όμως το συγκεκριμένο πρωτόκολλο έχει και ένα σημαντικό μειονέκτημα. Θεωρείται ιδιαίτερα βαρύ, δεδομένου του γεγονότος ότι χρειάζονται τουλάχιστον τρία πακέτα για την εγκαθίδρυση της σύνδεσης, πριν ακόμη μεταδοθεί οποιοδήποτε πακέτο δεδομένων. Επίσης, οι μηχανισμοί αξιοπιστίας που υλοποιεί το κάνουν ακόμη πιο βαρύ, πράγμα που έχει φυσικά σημαντικό αντίκτυπο στην ταχύτητα μετάδοσης δεδομένων.

Κάθε πακέτο που μεταφέρεται στο TCP αποτελείται από την επικεφαλίδα(source port, destination port κλπ), έναν πεδίο options για τον καθορισμό ειδικών επιλεγόμενων ρυθμίσεων και την ωφέλιμη πληροφορία όπως στο Σχήμα 13.



Bit offset	0–3	4–7	8–15	16–31
0	Source address			
32	Destination address			
64	Zeros		Protocol	TCP length
96	Source port			Destination port
128	Sequence number			
160	Acknowledgement number			
192	Data offset	Reserved	Flags	Window
224	Checksum			Urgent pointer
256	Options (optional)			
256/288+	Data			

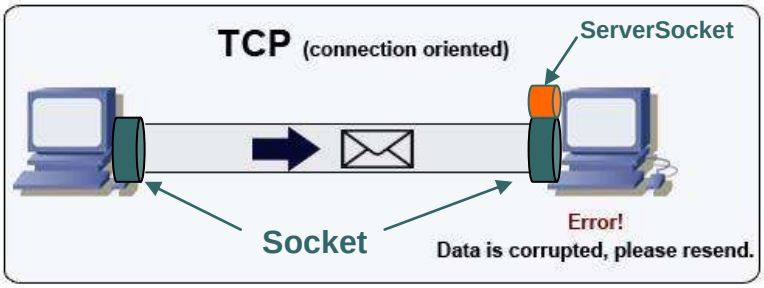
Σχήμα 13: Δομή TCP πακέτου

Το TCP είναι connection oriented, και η μεταφορά δεδομένων γίνεται από τις υποδοχές ροής(stream sockets) σε κάθε πλευρά μέσω εγκαθιδρυμένης σύνδεσης. Συνήθως η σύνδεση μέσω υποδοχών δεν είναι συμμετρική. Η μια διεργασία λειτουργεί ως διακομιστής(Server) και η άλλη ως πελάτης(Client).

Στην Java χειριζόμαστε τις συνδέσεις μέσω των αντίστοιχων κλάσεων των υποδοχών ροής σε κάθε πλευρά.

Το πακέτο java.net παρέχει την κλάση `ServerSocket`, η οποία υλοποιεί υποδοχές εξυπηρετητών (server sockets). Συνήθως ένας εξυπηρετητής εκτελείται σε μία συγκεκριμένη μηχανή, και έχει μια υποδοχή η οποία είναι συνδεδεμένη σε μία συγκεκριμένη τοπική θύρα. Η διεύθυνση της μηχανής και η τοπική θύρα είναι ευρέως γνωστές στους πελάτες. Ο εξυπηρετητής απλά περιμένει, αφουγκραζόμενος την υποδοχή για αιτήσεις σύνδεσης εκ μέρους πελατών.

Η κλάση `Socket`, η οποία περιέχεται επίσης στο πακέτο java.net, υλοποιεί τις υποδοχές πελατών (client sockets). Για τη δημιουργία μιας υποδοχής πελάτη είναι απαραίτητη η διεύθυνση IP (ή το όνομα) του υπολογιστή στον οποίο εκτελείται ο εξυπηρετητής με τον οποίον επιθυμεί να επικοινωνήσει ο πελάτης, καθώς και ο αριθμός της θύρας στην οποία είναι συνδεδεμένος ο εξυπηρετητής. Με βάση τα στοιχεία αυτά, ο πελάτης στέλνει μια αίτηση σύνδεσης στον εξυπηρετητή.



Σχήμα 14: Υποδοχές TCP

Αν όλα πάνε καλά, ο εξυπηρετητής κάνει αποδεκτή την αίτηση του πελάτη, με αποτέλεσμα τη δημιουργία μιας νέας υποδοχής η οποία είναι συνδεδεμένη σε μία διαφορετική θύρα. Η νέα αυτή υποδοχή είναι απαραίτητη, προκειμένου να μπορεί ο εξυπηρετητής να αφουγκράζεται για νέες αιτήσεις (στην αρχική υποδοχή), ενώ εξυπηρετεί τις ανάγκες του συγκεκριμένου πελάτη. Στην πλευρά του πελάτη, μετά την υποδοχή της αίτησής του από τον εξυπηρετητή, δημιουργείται μια υποδοχή, η οποία συνδέεται σε κάποια τοπική θύρα, μέσω της οποίας μπορεί πλέον να επικοινωνεί με τον εξυπηρετητή.

Δύο κατασκευαστές της κλάσης `ServerSocket` είναι οι εξής:

```
public ServerSocket (int port, int queue_length)
                                    throws IOException
```

δημιουργεί μια υποδοχή εξυπηρετητή, η οποία είναι συνδεμένη στη θύρα `port`. Αν η παράμετρος `port` έχει την τιμή 0, τότε η υποδοχή συνδέεται με οποιαδήποτε ελεύθερη θύρα. Για την υποδοχή αυτή δημιουργείται μια ουρά (αναμονής) εισερχόμενων αιτήσεων σύνδεσης, το μήκος της οποίας είναι `queue_length`. Οποιαδήποτε αίτηση αφιχθεί, ενώ η ουρά αυτή είναι γεμάτη, απορρίπτεται.

```
public ServerSocket (int port) throws IOException
```

διαφέρει από τον παραπάνω κατασκευαστή μόνο στο ότι η μέθοδος αυτή θέτει το μήκος της ουράς των εισερχόμενων αιτήσεων ίσο με 50.

Μερικές από τις μεθόδους της κλάσης `ServerSocket` είναι οι εξής:

```
public Socket accept () throws IOException
```

αφουγκράζεται για αιτήσεις σύνδεσης, και τις κάνει αποδεκτές, αν εννοείται αυτό είναι εφικτό. Η μέθοδος αυτή εμποδίζεται μέχρις ότου φτάσει μία αίτηση σύνδεσης στη θύρα με την οποία συνδέθηκε η υποδοχή κατά τη δημιουργία της,

```
public void close () throws IOException
```

κλείνει τη συγκεκριμένη υποδοχή. Τυχόν νήματα που έχουν εμποδιστεί στη μέθοδο **`accept`** παράγουν μία εξαίρεση τύπου `SocketException`.

Δύο κατασκευαστές της κλάσης `Socket` είναι οι εξής:

```
public Socket (InetAddress addr, int port) throws IOException
```

δημιουργεί μία υποδοχή ρεύματος, και τη συνδέει με τη θύρα `port` στη διεύθυνση `addr`. Μία εξαίρεση τύπου `IOException` παράγεται σε περίπτωση που συμβεί κάποιο σφάλμα εισόδου/εξόδου κατά τη δημιουργία της υποδοχής.

```
public Socket (String host, int port)
                        throws UnknownHostException, IOException
```

δημιουργεί μία υποδοχή ρεύματος, και τη συνδέει με τη θύρα `port` του ξένιου υπολογιστή με όνομα `host`. Μία εξαίρεση τύπου `IOException` παράγεται σε περίπτωση που συμβεί κάποιο σφάλμα εισόδου/εξόδου κατά τη δημιουργία της υποδοχής, ενώ μια εξαίρεση τύπου `UnknownHostException` παράγεται σε περίπτωση αποτυχίας επίλυσης του ονόματος `host`.

Μερικές από τις μεθόδους της κλάσης `Socket` είναι οι εξής:

```
public InputStream getInputStream () throws IOException
```

επιστρέφει ένα ρεύμα εισόδου για την ανάγνωση δεδομένων(bytes) από τη συγκεκριμένη υποδοχή ροής. Αν αυτή η υποδοχή ροής έχει ένα συσχετιζόμενο κανάλι

τότε το παραγόμενο input stream μεταβιβάζει όλες του τις λειτουργίες στο κανάλι. Αν κλείσει το επιστρεφόμενο `InputStream` θα κλείσει και η υποδοχή ροής.

```
public OutputStream getOutputStream () throws IOException
```

επιστρέφει ένα ρεύμα εξόδου για την εγγραφή δεδομένων(bytes) στη συγκεκριμένη υποδοχή ροής. Αν αυτή η υποδοχή ροής έχει ένα συσχετιζόμενο κανάλι τότε το παραγόμενο output stream μεταβιβάζει όλες του τις λειτουργίες στο κανάλι. Αν κλείσει το επιστρεφόμενο `OutputStream` θα κλείσει και η υποδοχή ροής.

```
public InputStream getInetAddress ()
```

επιστρέφει την απομακρυσμένη διεύθυνση στην οποία η υποδοχή ροής είναι συνδεμένη ή `null` εάν αυτή η υποδοχή δεν είναι συνδεμένη.

```
public void close () throws IOException
```

κλείνει τη συγκεκριμένη υποδοχή. Αν υπάρχει κάποιο νήμα το οποίο έχει εμποδιστεί για είσοδο/έξοδο στη συγκεκριμένη υποδοχή, τότε παράγει μία εξαίρεση τύπου `SocketException`.

Στο παρακάτω παράδειγμα, ο εξυπηρετητής δέχεται μια συμβολοσειρά από τον πελάτη και του την επιστρέφει αντεστραμμένη. Ο εξυπηρετητής είναι επαναληπτικός, δηλαδή ολοκληρώνει την εξυπηρέτηση μίας αίτησης προτού ασχοληθεί με την επόμενη αίτηση. Αν φτάσουν νέες αιτήσεις σύνδεσης προτού ολοκληρωθεί η εξυπηρέτηση της τρέχουσας, προστίθενται στην ουρά (αναμονής) της υποδοχής του εξυπηρετητή.

Αρχικά ο κώδικας του εξυπηρετητή:

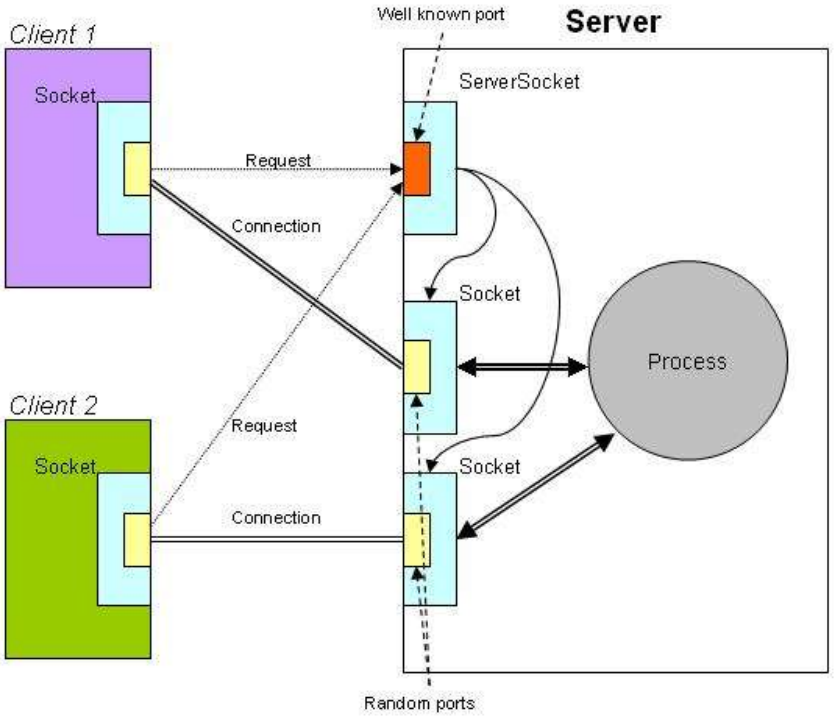
```
public class AntistrofosEksyphrethths
{
	public static void main(String[] args) throws IOException
	{
		// μια θύρα
		final int THYRA = 5000;

		// άκου για αιτήσεις σύνδεσης στη θύρα THYRA
		ServerSocket ss = new ServerSocket(THYRA);

		try
		{
			while(true)
			{
				Socket s = null;
				PrintWriter out = null;
				BufferedReader in = null;
				try
				{
					// περίμενε αίτηση σύνδεσης από πελάτη
					s = ss.accept();
					out = new PrintWriter(s.getOutputStream(),
					true);
					in = new BufferedReader(
					new InputStreamReader(s.getInputStream()));
					// διάβασε τη συμβολοσειρά
					String ti = in.readLine();
					// αντίστρεψέ την και επίστρεψε το
					// αποτέλεσμα στον πελάτη
				if(ti != null)
						out.println(antistrepse(ti));
				}
				finally
				{
					if(in != null) in.close();
					if(out != null) out.close();
					if(s != null) s.close();
				}
			}
		}
		finally
		{
			if(ss != null) ss.close();
		}
	}

	// παίρνει μία συμβολοσειρά και επιστρέφει την
	// αντίστοιχη αντεστραμμένη συμβολοσειρά
	private static String antistrepse(String s)
	{
		StringBuilder sb = new StringBuilder(s);
		return sb.reverse().toString();
	}
}
```

Ακολουθεί ο κώδικας του πελάτη:

```
public class AntistrofosPelaths
{
	public static void main(String[] args) throws IOException
	{
		// το όνομα οικοδεσπότη της μηχανής όπου
		// εκτελείται ο εξυπηρετητής
		final InetAddress DIEYTHYNSH_EKSYPHRETHTH =
			InetAddress.getByName("nefeli.teicrete.gr");
		// η θύρα όπου ο εξυπηρετητής ακούει για αιτήσεις
		final int THYRA_EKSYPHRETHTH = 5000;

		Socket s = null;
		BufferedReader in = null, stdin = null;
		PrintWriter out = null;
		try
		{
			// δημιούργησε την υποδοχή
			s = new Socket(DIEYTHYNSH_EKSYPHRETHTH,
				THYRA_EKSYPHRETHTH);
			out = new PrintWriter(s.getOutputStream(),
				true);
			in = new BufferedReader(new InputStreamReader
				(s.getInputStream()));
			stdin = new BufferedReader(new InputStreamReader
				(System.in));
			// ζήτα από το χρήστη τη συμβολοσειρά
			System.out.print("Αρχική συμβολοσειρά: ");
			String ti = stdin.readLine();
			if(ti != null)
			{
				// στείλε τη συμβολοσειρά στον εξυπηρετητή
				out.println(ti);
				// εμφάνισε στο χρήστη την απάντηση του
				// εξυπηρετητή (δηλαδή την αντεστραμμένη
				// συμβολοσειρά)
				System.out.println("Αντεστραμμένη " +
					"συμβολοσειρά: " + in.readLine());
			}
		}
		finally
		{	if(stdin != null) stdin.close();
			if(in != null) in.close();
			if(out != null) out.close();
			if(s != null) s.close();
		}
	}
}
```

4.2. Μηχανισμός Client - Server

Υλοποιώντας σε Java μια Client-Server εφαρμογή στο TCP(Σχήμα 15), η διεργασία στον εξυπηρετητή (server process) θα πρέπει αρχικά να αφουγκράζεται για αιτήσεις για σύνδεση σε μία συγκεκριμένη πόρτα «γνωστή» στους πελάτες(clients). Έτσι δημιουργούμε ένα `ServerSocket` με συγκεκριμένο αριθμό πόρτας(well known port). Στο πελάτη θα χρησιμοποιήσουμε ένα `Socket` μέσω του οποίου θα γίνει και η αρχική αίτηση για σύνδεση με τον εξυπηρετητή καθώς και η υπόλοιπη επικοινωνία μαζί του μετά την επιτυχή σύνδεση. Στο πελάτη δε χρειάζεται να ορίσουμε για το `Socket` συγκεκριμένη τοπική πόρτα. Αφήνουμε το σύστημα να δεσμεύσει την πρώτη ελεύθερη πόρτα που υπάρχει διαθέσιμη και μέσω των μηνυμάτων που θα σταλθούν στον εξυπηρετητή η διεύθυνση και η πόρτα του πελάτη θα γίνει γνωστή σε αυτόν και θα γίνει κατάλληλα η σύνδεση μεταξύ τους. Στον εξυπηρετητή το `ServerSocket` μας επιστρέφει ένα `Socket` (αντίστοιχο με αυτό του πελάτη) για κάθε επιτυχημένη σύνδεση με ένα πελάτη. Επομένως ο εξυπηρετητής έχει τόσα `Socket` όσοι και οι συνδεμένοι πελάτες. Ωστόσο για να διαχειριστεί πολλούς συνδεδεμένους πελάτες ταυτόχρονα θα πρέπει να γίνει χρήση νημάτων(`Thread`). Μετά την σύνδεση η αποστολή των μηνυμάτων γίνεται μέσω των υποδοχών μέσα από κανάλια ροής δεδομένων. Ποίο συγκεκριμένα κάθε `Socket` διαθέτει ένα εισερχόμενο(input) και ένα εξερχόμενο(output) κανάλι ροής(stream). Για την διαχείριση των input/output streams του κάθε `Socket` είναι απαραίτητη η χρήση βοηθητικών κλάσεων ανάγνωσης δεδομένων από κανάλια ροής όπως οι `BufferedReader`, `InputStreamReader`, `PrintWriter`, κλπ.



Σχήμα 15: Μηχανισμός TCP Client-Server

Παρακάτω παρατίθεται κώδικας σε Java για μονόδρομη επικοινωνία όπου ο πελάτης(client) στέλνει μηνύματα στον εξυπηρετητή(server) και αυτός τα εκτυπώνει στην έξοδο. Σε περίπτωση που θέλουμε η επικοινωνία να είναι αμφίδρομη ακολουθούμε αντίστοιχες διαδικασίες(receive, send) και στις δύο πλευρές(client-server), ωστόσο θα χρειαστεί να υλοποιηθούν οι διαδικασίες αυτές σε νήματα (`Thread`) για να μπορούν να εκτελεστούν (ψευδο)παράλληλα σε κάθε πλευρά. Επιπλέον αν ο εξυπηρετητής πρέπει να επικοινωνεί με πολλούς πελάτες ταυτόχρονα θα πρέπει να υλοποιήσει το κάθε `Socket`(συγκεκριμένος πελάτης) σε ξεχωριστό νήμα.

Αρχικά ο κώδικας του εξυπηρετητή:

```
public class Server {

	private int serverPort=13000;

	private ServerSocket serverSocket;

	public Server(){
		try {
			serverSocket = new ServerSocket(serverPort);

			System.out.println("Waiting...");

			Socket socket=serverSocket.accept();

			BufferedReader input = new BufferedReader(new
				InputStreamReader(socket.getInputStream()));

			while (true) {
				String receivedData = input.readLine();
				System.out.println("ReceivedData: " +
receivedData);
			}

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

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

Ακολουθεί ο κώδικας του πελάτη:

```
public class Client {

	private Socket socket;

	private static InetAddress serverAddress;

	public Client() {
		try {
			socket = new Socket(serverAddress, 13000);
		} catch (UnknownHostException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	private void sendData() {

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

		System.out.println("Type your messages");

		while (true) {
			try {
				String data = br.readLine();
				PrintWriter output = new
					PrintWriter(socket.getOutputStream(),
					true);
				output.println(data);
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}

	public static void main(String[] args) {
		if (args.length != 1) {
			System.out.println("Error: forgot or more than once
<server IP>");
			return;
		}

		String serverIP = args[0];
		try {
			serverAddress = InetAddress.getByName(serverIP);
		} catch (UnknownHostException e) {
			e.printStackTrace();
		}

		Client client = new Client();
		client.sendData();
	}
}
```

4.3. Ανάπτυξη TCP εφαρμογής

Να υλοποιηθεί client – server μηχανισμός σύμφωνα με το πρωτόκολλο δικτύου TCP κατά τον οποίο ο client θα διαβάζει μηνύματα από το πληκτρολόγιο(ή textfield σε περίπτωση που τον υλοποιήσετε με διεπαφή) και θα τα στέλνει στον server. Τα μηνύματα που θα αποστέλλονται θα είναι τυποποιημένα της μορφής:

a) Login <username>
b) Message <message>
c) Users
d) Logout

Ο server θα πράττει, αναλόγως τα προθέματα, ως εξής:

a) Θα αποθηκεύει τα απαραίτητα στοιχεία του χρήστη σε μια λίστα με ενεργούς χρήστες.
β) Θα στέλνει το <message> σε όλους τους ενεργούς χρήστες.
c) Θα επιστρέφει στον αποστολέα την λίστα με τους ενεργούς χρήστες (usernames).
d) Θα σβήνει το χρήστη από την λίστα των ενεργών χρηστών.

Προσέξτε τα παρακάτω

Μόνο στον server χρειάζεται να δηλωθεί port στην οποία θα «ακούει» για requested connections (ServerSocket). Η επικοινωνία γίνεται έπειτα μέσω των sockets.

Ο Server θα πρέπει να κρατά ανοιχτά τα connections με κάθε client, γεγονός που σημαίνει ότι πρέπει να δημιουργεί ένα Thread για κάθε Client που συνδέεται, μέσω του οποίου θα τον εξυπηρετεί.

Η μεταφορά των δεδομένων γίνεται μέσω των streams(input,output) των sockets σε κάθε πλευρά.

5. Μετανάστευση εφαρμογών με Σειριακοποίηση

5.1. Εισαγωγή

Σειριακοποίηση(serialization) είναι η διαδικασία με την οποία ένα αντικείμενο αποθηκεύεται σε ένα μέσο αποθήκευσης(σκληρός, μνήμη κλπ) η μεταφέρεται μέσα από ένα δίκτυο, σε δυαδική διάταξη. Από αυτή τη διάταξη, έπειτα, το αντικείμενο μπορεί να δημιουργηθεί πάλι με ταυτόσημη δομή και στην τελευταία του κατάσταση(ουσιαστικά ένας κλώνος).

Οι υποδοχές ρευμάτων δεδομένων υλοποιούν μια αμφίδρομη αξιόπιστη ροή δυφιοσυλλαβών ανάμεσα στα άκρα που επικοινωνούν. Στο χαμηλότερο επίπεδο, τα δύο άκρα μπορούν να παραλαμβάνουν και να στέλνουν δυφιοσυλλαβές μέσω των κλάσεων ρευμάτων `InputStream` και `OutputStream` της Java (ή των τάξεων που τις επεκτείνουν) με τον ίδιο τρόπο που θα χρησιμοποιούσαν ένα οποιοδήποτε ρεύμα εισόδου / εξόδου. Σε ελαφρά υψηλότερο επίπεδο αφαίρεσης, έχουμε τις κλάσεις ρευμάτων `DatalnputStream` και `DataOutputStream` του πακέτου java.io που υποστηρίζουν τη μεταφορά τιμών των πρωταρχικών (ενυπάρχοντων) τύπων, καθώς και συμβολοσειρών. Έτσι ο προγραμματιστής δεν χρειάζεται να ασχοληθεί με τον τρόπο αναπαράστασης των τιμών αυτών των τύπων κατά τη μεταφορά τους μέσω μιας υποδοχής ή, γενικότερα, μέσω ενός ρεύματος δεδομένων.

Σε ακόμα υψηλότερο επίπεδο αφαίρεσης, έχουμε τις κλάσεις ρευμάτων `ObjectlnputStream` και `ObjectOutputStream` του ίδιου πακέτου που υποστηρίζουν τη μεταφορά ολόκληρων αντικειμένων. Όμως κάποιο αντικείμενο μπορεί να αναφέρεται σε άλλα αντικείμενα και αυτά με τη σειρά τους σε άλλα κ.ο.κ., με αποτέλεσμα να δημιουργούν ένα γράφο αναφορών. Με χρήση της μεθόδου writeObject μπορούμε να γράψουμε σε ένα `ObjectOutputStream` όλες τις δυφιοσυλλαβές που συνθέτουν ένα αντικείμενο, συμπεριλαμβανομένων και των αντικειμένων στα οποία αυτό (άμεσα ή έμμεσα) αναφέρεται.

Η διαδικασία μετατροπής της εσωτερικής αναπαράστασης των αντικειμένων σε ένα ρεύμα δυφιοσυλλαβών ονομάζεται σειριακοπονηση (serialization), ενώ η αντίστροφη διαδικασία ονομάζεται αποσειριακοποίηση (deserialization). Όταν οι δυφιοσυλλαβές που κωδικοποιούν ένα σειριακοποιημένο γράφο αντικειμένων διαβαστούν από τη μέθοδο readObject της κλάσης `ObjectlnputStream`, αποσειριακοποιούνται, δηλαδή μετατρέπονται σε ένα γράφο αντικειμένων που είναι ισοδύναμος με τον αρχικό γράφο.

Με χρήση των μεθόδων αυτών των τάξεων, ολόκληρα αντικείμενα μπορούν να γραφτούν σε ένα αρχείο για να ανακτηθούν αργότερα ή να τοποθετηθούν σε ένα αντικείμενο τύπου `StringBuffer` ή να γραφτούν σε μια υποδοχή για να σταλούν μέσω του δικτύου. Για παράδειγμα, οι παρακάτω εντολές γράφουν ένα αντικείμενο της κλάσης S σε ένα ρεύμα `ObjectOutputStream`, το οποίο με τη σειρά του στέλνει το αντικείμενο στο υποκείμενο ρεύμα εισόδου/εξόδου:

```
// δημιούργησε ένα αντικείμενο της κλάσης S
S s = new S() ;
// δημιούργησε ένα ρεύμα εξόδου προς τον προορισμό του
// αντικειμένου
OutputStream out = ... ;
// περιτύλιξέ το με ένα ρεύμα τύπου
ObjectOutputStream ObjectOutputStream out = new ObjectOutputStream(out);
// γράψε το αντικείμενο στο ρεύμα αυτό
oout.writeObject (s) ;
```

Ο παραλήπτης μπορεί εξίσου εύκολα να αναδομήσει το αντικείμενο που του //έχει σταλεί, με εντολές της μορφής:

```
// δημιούργησε ένα ρεύμα εισόδου από την πηγή του
// αντικειμένου
InputStream in = ... ;
// περιτύλιξέ το με ένα ρεύμα τύπου
ObjectInputStream ObjectInputStream oin = new ObjectInputStream(in);
// διάβασε ένα αντικείμενο από το ρεύμα αυτό
S s = (S) oin.readObject();
```

Εννοείται ότι οι εντολές αυτές πρέπει να βρίσκονται μέσα σε μπλοκ try, καθώς και ότι πρέπει να υπάρχουν τα κατάλληλα μπλοκ catch για να συλλαμβάνουν τις εξαιρέσεις τους.

Στην Java, οι κλάσεις των αντικειμένων που επιθυμούμε να σειριακοποιηθούν πρέπει να υλοποιούν τη διεπαφή `Serializable` του πακέτου java.io. Οι περισσότερες κλάσεις των πακέτων java.lang και java.util την υλοποιούν. Τυχόν προσπάθεια σειριακοποίησης ενός αντικειμένου που αναφέρεται σε μη σειριακοποιήσιμα αντικείμενα ή ενός αντικειμένου, η κλάση του οποίου δεν υλοποιεί τη διεπαφή `Serializable`, προξενεί τη ρίψη μιας εξαίρεσης τύπου java.io.NotSerializableException.

Ένας κατασκευαστής της κλάσης `ObjectOutputStream` είναι ο εξής:

```
public ObjectOutputStream (OutputStream out)
                                    throws IOException
```

δημιουργεί ένα `ObjectOutputStream` το οποίο εγγράφει στο συγκεκριμένο `OutputStream out`.

Μερικές από τις μεθόδους της κλάσης `ObjectOutputStream` είναι οι εξής:

```
public final void writeObject (Object out) throws IOException
```

εγγράφει το συγκεκριμένο `Object obj` στο `ObjectOutputStream`.

```
public void flush () throws IOException
```

αδειάζει τη ροή. Αυτή η μέθοδος εγγράφει οποιανδήποτε bytes βρίσκονται στο buffer και τα προωθεί άμεσα στη ροή.

Ένας κατασκευαστής της κλάσης `ObjectInputStream` είναι ο εξής:

```
public ObjectInputStream (InputStream in)
                                    throws IOException
```

δημιουργεί ένα `ObjectInputStream` το οποίο διαβάζει από το συγκεκριμένο `InputStream in`.

Μερικές από τις μεθόδους της κλάσης `ObjectInputStream` είναι οι εξής:

```
public final Object readObject ()
                        throws IOException, ClassNotFoundException
```

διαβάζει ένα `Object` από το `ObjectInputStream` και το επιστρέφει.

5.2. Ανάπτυξη μηχανισμού σειριακοποίησης εφαρμογής

Να υλοποιηθεί μηχανισμός, σύμφωνα με το πρωτόκολλο δικτύου TCP, κατά τον οποίον θα γίνεται αποστολή και λήψη μιας εφαρμογής. Το γραφικό περιβάλλον του μηχανισμού θα αποτελείται από ένα παράθυρο με ένα *textfield* και 2 *buttons*. Στο *textfield* θα δηλώνεται η διεύθυνση (ip) του παραλήπτη, το ένα *button* θα χρησιμεύει στην προεπισκόπηση(view) και επεξεργασία της εφαρμογής και το άλλο στην αποστολή(send) της εφαρμογής. Η εφαρμογή αυτή θα είναι μια φόρμα συμπλήρωσης στοιχείων του χρήστη και θα αποτελείται από μερικά *labels* και *textfields* καθώς και ενός *button* για τον καθαρισμό(clear) των πεδίων. Ο μηχανισμός θα δέχεται παράλληλα και εφαρμογές-φόρμες από άλλους χρήστες στο δίκτυο οι οποίοι επίσης τον χρησιμοποιούν. Κάθε φορά που δέχεται μια τέτοια εφαρμογή θα την εμφανίζει άμεσα στην οθόνη του υπολογιστή με τίτλο την διεύθυνση του αποστολέα.

Προσοχή στα παρακάτω

Ο μηχανισμός λειτουργεί και σαν server (για την αποδοχή εφαρμογών από τρίτους) και σαν Client (για την αποστολή της δικής του εφαρμογής) που σημαίνει ότι υλοποιεί και ServerSocket με Threads και απλό Socket.

Θα αποστέλλεται ολόκληρη η φόρμα(frame) με χρήση Serialization.

Κάνετε implement το interface Serializable αν και όπου χρειάζεται.

Χρησιμοποιήστε τις κλάσεις ObjectOutputStream και ObjectInputStream για την διαχείριση των ροών των μεταφερόμενων αντικειμένων.

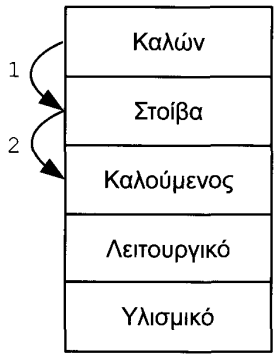
Η εφαρμογή αφού αποσταλεί θα πρέπει να εμφανίζεται στον παραλήπτη με την τελευταία της κατάσταση και να διαθέτει όλη της την λειτουργικότητα.

6. Απομακρυσμένη κλήση διαδικασιών

6.1. Εισαγωγή

Αν και πολλά κατανεμημένα συστήματα χρησιμοποιούν ρητή ανταλλαγή μηνυμάτων ανάμεσα στις διεργασίες, η χρήση των κλήσεων send και receive καταργεί τη διαφάνεια του συστήματος και καθιστά εμφανή τη διάκριση ανάμεσα στην τοπική επικοινωνία και την επικοινωνία μέσω δικτύου. Για να αποφύγουμε τη διαφοροποίηση ανάμεσα στην τοπική και τη δικτυακή επικοινωνία, συχνά χρησιμοποιούμε το μοντέλο της κλήσης απομακρυσμένων διαδικασιών (remote procedure call, RPC). Αυτό το μοντέλο είναι μια λογική αφαίρεση υψηλότερου επιπέδου, η οποία επιτρέπει στις διεργασίες που εκτελούνται σε κάποια μηχανή να καλούν διαδικασίες οι οποίες εκτελούνται σε άλλες μηχανές. Όταν μια διεργασία καλέσει μια διαδικασία που βρίσκεται σε άλλη μηχανή, η καλούσα διεργασία εμποδίζεται μέχρι να τελειώσει η εκτέλεση της καλούμενης διαδικασίας στην άλλη μηχανή και να της επιστραφούν τα αποτελέσματα της κλήσης. Έτσι οι απομακρυσμένες κλήσεις αποκρύπτουν από τον προγραμματιστή τις λεπτομέρειες της επικοινωνίας μεταξύ διαφορετικών μηχανών και διεργασιών.

Για να κατανοήσουμε πώς υλοποιείται το μοντέλο RPC, θα εξετάσουμε πρώτα τον τρόπο υλοποίησης των τοπικών κλήσεων και μετά θα δούμε τις αλλαγές που επιβάλλει η απομακρυσμένη λειτουργία. Στο Σχήμα 16 φαίνεται ο τρόπος υλοποίησης μιας τοπικής κλήσης διαδικασίας. Η κλητική ακολουθία της διαδικασίας αρχικά δεσμεύει χώρο στη στοίβα για τις τυπικές παραμέτρους, καθώς και για το αποτέλεσμα στην περίπτωση συναρτήσεων.

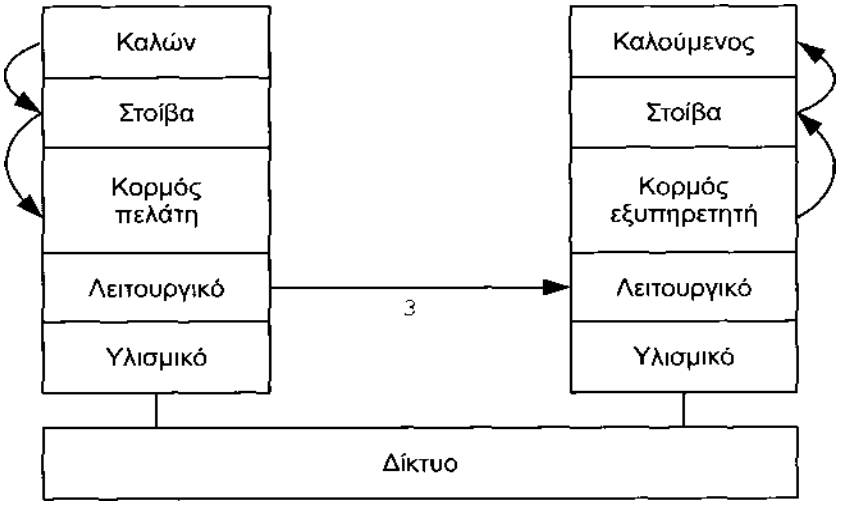


Σχήμα 16: Τοπική κλήση διαδικασίας.

Στη συνέχεια εκχωρεί τις πραγματικές παραμέτρους στις θέσεις που δέσμευσε, με τρόπο που εξαρτάται από τη χρησιμοποιούμενη μέθοδο μεταβίβασης παραμέτρων, και μεταφέρει τον έλεγχο στην καλούμενη διαδικασία. Η καλούμενη διαδικασία διαβάζει τις παραμέτρους της από τη στοίβα και εκτελεί τον κώδικά της. Όταν ο έλεγχος επιστραφεί στην καλούσα διαδικασία, τότε ελευθερώνεται ο χώρος που είχε δεσμευθεί στη στοίβα, αφού αντιγραφεί το αποτέλεσμα της κλήσης (στην περίπτωση των συναρτήσεων) και πιθανόν οι παράμετροι που έχουν μεταβληθεί. Ενδιάμεσα η καλούμενη διαδικασία μπορεί να καλέσει άλλες διαδικασίες, ή ακόμα και τον εαυτό της, πράγμα που δεν απασχολεί την καλούσα διαδικασία.

Η λογική αφαίρεση που παρέχει το μοντέλο RPC αποκρύπτει από την καλούσα διαδικασία το ότι η καλούμενη διαδικασία εκτελείται σε μια διαφορετική μηχανή, αλλά και αντίστροφα, αποκρύπτει από την καλούμενη διαδικασία το ότι η καλούσα διαδικασία εκτελείται σε μια άλλη μηχανή. Ο στόχος είναι η καλούσα και η καλούμενη διαδικασία να είναι ακριβώς ίδιες στον προγραμματισμό, είτε επικοινωνούν τοπικά είτε μέσω δικτύου. Αυτό επιτυγχάνεται με την απόκρυψη των λεπτομερειών της κλήσης μέσω μιας κατάλληλης βιβλιοθήκης, όπως γίνεται και με τις κλήσεις του συστήματος. Για παράδειγμα, όταν διαβάζουμε στοιχεία από ένα αρχείο, αν και τελικά θα χρειαστεί να εκτελεστεί από τον πυρήνα η κλήση read του συστήματος, το πρόγραμμά μας αρκεί να καλέσει τη διαδικασία read της βιβλιοθήκης, η οποία αναλαμβάνει την εκτέλεση της κλήσης του συστήματος.

Με αντίστοιχο τρόπο, όταν η διαδικασία που πρόκειται να κληθεί βρίσκεται σε μια απομακρυσμένη μηχανή, στη θέση της απομακρυσμένης διαδικασίας καλούμε μια διαδικασία της βιβλιοθήκης με το ίδιο όνομα η οποία είναι γνωστή ως κορμός



Σχήμα 17: Κλήση απομακρυσμένης διαδικασίας.

πελάτη (client stub) , όπως φαίνεται στο Σχήμα 17. Αυτή η τοπική διαδικασία καλείται με την ίδια κλητική ακολουθία όπως η απομακρυσμένη, δηλαδή με τις ίδιες παραμέτρους. Όμως, αντί να χειριστεί η ίδια την κλήση, τοποθετεί τις παραμέτρους της κλήσης σε ένα μήνυμα και καλεί τη διαδικασία send, ζητώντας έτσι από τον πυρήνα να στείλει αυτό το μήνυμα προς τον εξυπηρετητή. Κατόπιν, ο κορμός πελάτη καλεί τη διαδικασία receive, η οποία προξενεί τον εμποδισμό του μέχρι να έρθει η απάντηση από τον εξυπηρετητή.

Ο πυρήνας του πελάτη στέλνει το μήνυμα στον πυρήνα του εξυπηρετητή, ο ο-ποίος, αφού το παραλάβει, το μεταβιβάζει σε μια διαδικασία κορμού εξυπηρετητή (server stub). Ο κορμός εξυπηρετητή είναι η ομόλογη οντότητα του κορμού πελάτη, δηλαδή ανακτά τις παραμέτρους από το μήνυμα και καλεί τοπικά τη διαδικασία στον εξυπηρετητή με το συνηθισμένο τρόπο, προσθέτοντας τις παραμέτρους στη στοίβα και εκτελώντας την κλήση. Η καλούμενη διαδικασία εκτελεί τα καθήκοντά της χωρίς να αντιλαμβάνεται ότι έχει κληθεί από απόσταση. Όταν ο έλεγχος επιστρέψει στον κορμό του εξυπηρετητή, αυτός τοποθετεί τα αποτελέσματα σε ένα μήνυμα απάντησης και καλεί τη διαδικασία send για να τα επιστρέψει στον κορμό του πελάτη.

Όταν το μήνυμα επιστρέψει στον πυρήνα της μηχανής του πελάτη, αντιγράφεται στο χώρο διευθύνσεων της διεργασίας που το περιμένει, δηλαδή στον κορμό του πελάτη, και ο κορμός του πελάτη ενημερώνεται ότι η κλήση receive έχει ολοκληρωθεί. Ο κορμός του πελάτη ανακτά τα αποτελέσματα από το μήνυμα που έλαβε και τα επιστρέφει με το συνηθισμένο τρόπο στην καλούσα διαδικασία, δηλαδή μέσω της στοίβας, χωρίς η καλούσα διαδικασία να καταλάβει ότι η διαδικασία που κάλεσε εκτελέσθηκε απομακρυσμένα. Έτσι οι λεπτομέρειες μεταβίβασης των μηνυμάτων ανάμεσα στις μηχανές αποκρύπτονται από τις διαδικασίες του χρήστη χάρη στις διαδικασίες-κορμούς της βιβλιοθήκης.

Με τον παραπάνω τρόπο μπορούμε να κάνουμε (σχεδόν) οποιαδήποτε διαδικασία διαθέσιμη για απομακρυσμένες κλήσεις, αρκεί να γράψουμε τους κατάλληλους κορμούς πελάτη και εξυπηρετητή.

6.2. Σύστημα RMI(Remote Method Invocation)

Το RMI είναι ένα Java API που λειτουργεί σύμφωνα με το μοντέλο RPC(remote procedure calls-απομακρυσμένη κλήση διαδικασιών).

Το RMI επιτρέπει την κλήση μεθόδων οι οποίες βρίσκονται απομακρυσμένα, μοιράζοντας πόρους και φόρτο εργασίας μεταξύ συστημάτων. Αντίθετα με άλλα συστήματα απομακρυσμένης εκτέλεσης διεργασιών όπου μόνο απλές η συγκεκριμένες δομές δεδομένων μεταφέρονται μεταξύ των μεθόδων, το RMI επιτρέπει την χρήση οποιουδήποτε αντικειμένου ακόμα και αν είναι η πρώτη φορά που ένα σύστημα έρχεται σε επαφή μαζί του(δυναμικό φόρτωμα νέων αντικειμένων).

Οι RMI εφαρμογές συχνά ακολουθούν το μοντέλο client-server. Ο εξυπηρετητής(server) δημιουργεί μερικά απομακρυσμένα αντικείμενα, δημιουργεί αναφορές προς αυτά ώστε να είναι προσβάσιμα και περιμένει τους πελάτες(clients) να καλέσουν μεθόδους σε αυτά τα αντικείμενα. Ένας πελάτης αποκτά μία απομακρυσμένη αναφορά προς ένα η περισσότερα απομακρυσμένα αντικείμενα στον εξυπηρετητή και έπειτα καλεί μεθόδους σε αυτά. Το RMI API περιλαμβάνει όλο το μηχανισμό με τον οποίο ένας εξυπηρετητής και ένας πελάτης επικοινωνούν και ανταλλάσσουν δεδομένα μεταξύ τους.

Υλοποιώντας μία RMI εφαρμογή, πρώτα πρέπει να οριστεί μία διασύνδεση(interface) στην οποία μόνο θα δηλωθούν οι μέθοδοι που θα έχει το απομακρυσμένο αντικείμενο. Αν χρειαζόμαστε πολλά απομακρυσμένα αντικείμενα υλοποιούμε τόσες διασυνδέσεις όσα και τα αντικείμενα. Έπειτα θα πρέπει να υλοποιηθεί το κάθε απομακρυσμένο αντικείμενο που θα διαθέτει ο εξυπηρετητής το οποίο θα πρέπει να κληρονομεί(extends) τη κλάση `UnicastRemoteObject` του πακέτου java.rmi.server και να υλοποιεί(implements) την αντίστοιχη-κατάλληλη διασύνδεση με τις δηλωμένες μεθόδους έτσι ώστε αυτό το αντικείμενο να υλοποιεί στην ουσία την λειτουργικότητα τους. Επόμενο βήμα είναι η δημιουργία του εξυπηρετητή ο οποίος θα δηλώνει το κάθε απομακρυσμένο αντικειμένο στην RMI Registry(registry που παρέχει το RMI API) με ένα συγκεκριμένο αναγνωριστικό όνομα. Στο τέλος δημιουργούμε τον πελάτη στον οποίο γίνεται η κλήση των απομακρυσμένων αντικειμένων(καλώντας με το αναγνωριστικό όνομα από την RMI Registry του εξυπηρετητή) και κατ επέκταση των απομακρυσμένων μεθόδων-διαδικασιών του.

Παρακάτω παρατίθεται κώδικας για την κλήση των απομακρυσμένων μεθόδων getMessage(String user) και getTime() του απομακρυσμένου αντικειμένου `HelloServiceImpl`:

Αρχικά ο κώδικας της διασύνδεσης `HelloService` με τις δηλωμένες μεθόδους:

```
public interface HelloService extends Remote {

    //Δήλωση της μεθόδου <<getMessage>>.
    public String getMessage(String username) throws RemoteException;

    //Δήλωση της μεθόδου <<getTime>>.
    public Date getTime() throws RemoteException;
}
```

Ο κώδικας του απομακρυσμένου αντικειμένου `HelloServiceImpl` που υλοποιεί την διασύνδεση με τις δηλωμένες μεθόδους άρα την λειτουργικότητα αυτών:

```
public class HelloServiceImpl extends UnicastRemoteObject
                                    implements HelloService {

    //Κατασκευαστής του απομακρυσμένου αντικειμένου.
    //Καλεί τον κατασκευαστή της υπερκλάσης(UnicastRemoteObject).
    public HelloServiceImpl() throws RemoteException {
        super();
    }
    //Υλοποιεί την λειτουργικότητα της μεθόδου <<getMessage>>.
    public String getMessage(String username) throws RemoteException {
        return "Hello " + username + " :) !!!";
    }
    //Υλοποιεί την λειτουργικότητα της μεθόδου <<getTime>>.
    public Date getTime() throws RemoteException {
        return new Date();
    }
}
```

Ο κώδικας του εξυπηρετητή `RMIServer` που δημιουργεί και δηλώνει στην RMI Registry το απομακρυσμένο αντικείμενο `HelloServiceImpl`:

```
public class RMIServer {

    //Ο κατασκευαστής του RMI Server.
    public RMIServer() {

        try {

            //Δημιουργία του απομακρυσμένου αντικειμένου με τις μεθόδους.
            HelloServiceImpl hsi = new HelloServiceImpl();

            //Εντοπισμός της RMI Registry και αναφορά σε αυτή(registry).
            Registry registry = LocateRegistry.getRegistry();

            try {

                //Δήλωση(bind) του απομακρυσμένου αντικειμένου στην RMI Registry
                //με αναγνωριστικό όνομα «HelloService».
                registry.bind("HelloService", hsi);

            } catch (AlreadyBoundException ex) {
                ex.printStackTrace();
            } catch (AccessException ex) {
                ex.printStackTrace();
            }

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

    }

    //Κύρια συνάρτηση. Δημιουργία του RMIServer.
    public static void main(String[] args) {
        new RMIServer();
    }
}
```

Ο κώδικας του πελάτη `RMIClient` που καλεί το απομακρυσμένο αντικείμενο με αναγνωριστικό όνομα «HelloService» του εξυπηρετητή και εν συνεχεία τις μεθόδους-διαδικασίες:

```
public class RMIClient {
    //Η IP διεύθυνση του εξυπηρετητή.
    private String serverAddress = "127.0.0.1";

    //Ο κατασκευαστής του RMI Client.
    public RMIClient() {

        try {
            //Εντοπισμός της RMI Registry του εξυπηρετητή και
            //αναφορά σε αυτή(registry).
            Registry registry = LocateRegistry.getRegistry(serverAddress);

            try {
               //Αναζήτηση απομακρυσμένου αντικειμένου με
               //αναγνωριστικό «HelloService»
               HelloService hs = (HelloService) registry.lookup("HelloService");

               //Κλήση της απομακρυσμένης μεθόδου «getMessage».
               //Επιστρέφει το αποτέλεσμα εκτέλεσης αυτής στον εξυπηρετητή.
               String message = hs.getMessage("Dimitris");

               System.out.println(message);

            } catch (NotBoundException ex) {
                ex.printStackTrace();
            } catch (AccessException ex) {
                ex.printStackTrace();
            }


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


    }
    //Κύρια συνάρτηση. Δημιουργία του RMI Client.
    public static void main(String[] args) {
        new RMIClient();
    }
}
```

6.3. Ανάπτυξη RMI συστήματος

Να υλοποιηθεί RMI Client – Server μηχανισμός κατά τον οποίο οι χρήστες μπορούν να κάνουν κλήση απομακρυσμένων διεργασιών. Ο Server θα διαθέτει δύο «απομακρυσμένα αντικείμενα» με διάφορες μεθόδους σε καθένα από αυτά. Το πρώτο θα αφορά πράξεις(addition, subtraction, multiplication, division) δύο ακεραίων και το άλλο επεξεργασία δύο συμβολοσειρών(concat, equals). Ο Client θα είναι μία διεπαφή η οποία θα αποτελείται από τουλάχιστον, 3 textfield, ένα για την εισαγωγή της διεύθυνσης του server και δύο για την εισαγωγή των δεδομένων(ακέραιοι η συμβολοσειρές), 1 textarea, για την εμφάνιση των αποτελεσμάτων και 2 buttons, ένα για την κλήση των μεθόδων του 1ου «απομακρυσμένου αντικείμενου» και ένα για την κλήση των μεθόδων του 2ου. Έτσι ο χρήστης συμπληρώνοντας την διεύθυνση του Server και τα πεδία με τα δεδομένα αναλόγως ποίο κουμπί θα πατήσει θα εκτελούνται μία-μία οι μέθοδοι του επιλεγμένου «αντικειμένου» και θα του επιστρέφονται τα δεδομένα στο textarea όπως στις περιπτώσεις στο παρακάτω παράδειγμα:

a) //δεδομένα ‘5’ και ‘2’
5+2 ➔ 7
5–2 ➔ 3
5*2 ➔ 10
5/2 ➔ 2,5

b) //δεδομένα ‘geia’ και ‘geia’
concat ➔ geiageia
equals ➔ true

Προσοχή στα παρακάτω

Θα πρέπει να φτιάξετε ένα Interface με τις μεθόδους για κάθε «απομακρυσμένο αντικείμενο» το οποίο θα κάνει extends to java.rmi.Remote ενώ κάθε μέθοδος θα πρέπει να κάνει throws RemoteException.

Χρησιμοποιήστε τα ονόματα addition, subtraction, multiplication, division για τις μεθόδους του 1ου «αντικειμένου» και concat, equals για τις μεθόδους του 2ου.

Η concat θα παίρνει δύο string και θα τοποθετεί το δεύτερο στο τέλος του πρώτου (επιστρέφει string) και η equals θα παίρνει δύο string και θα τα συγκρίνει για ισότητα (επιστρέφει boolean).

Θα πρέπει να υλοποιήσετε τα «απομακρυσμένα αντικείμενα» κάνοντας implements τα αντίστοιχα Interfaces.

Χρησιμοποιήστε την κλάση Registry για να κάνετε bind και lookup τα «αντικείμενα».
Registry registry = LocateRegistry.getRegistry();

Στον client χρησιμοποιούμε τα interfaces για να γίνει αναφορά προς τα «απομακρυσμένα αντικείμενα» και να κάνουμε την κλήση των μεθόδων.

Πριν τρέξετε τον server θα πρέπει να εκκινήσετε την Rmi Registry πληκτρολογώντας στο command line> start rmiregistry.

7. Προδιαγραφές απομακρυσμένων διαδικασιών

7.1. Εισαγωγή

Αφού καθοριστούν οι κανόνες επικοινωνίας σε ένα σύστημα RPC, όπως το πρωτόκολλο επικοινωνίας που χρησιμοποιείται, τον τρόπο αναπαράστασης των παραμέτρων και τη σειρά μεταβίβασής τους, τυποποιείται σε μεγάλο βαθμό η υλοποίηση των κορμών πελάτη και εξυπηρετητή. Αυτό σημαίνει ότι μπορούμε να χρησιμοποιήσουμε έτοιμες διαδικασίες της βιβλιοθήκης για την πρόταξη των διαφόρων τύπων των παραμέτρων, καθώς και για την αποστολή και λήψη μηνυμάτων. Για παράδειγμα, η βιβλιοθήκη μπορεί να περιέχει διαδικασίες για πρόταξη ακεραίων και πραγματικών αριθμών, καθώς και χαρακτήρων και συμβολοσειρών. Στην πραγματικότητα, το μόνο στοιχείο που διαφοροποιεί τους κορμούς διαφόρων διαδικασιών μεταξύ τους είναι οι τυπικές προδιαγραφές, ή υπογραφές (signatures), των διαδικασιών που παρέχει ο εξυπηρετητής και τις οποίες μπορεί να καλέσει ο πελάτης. Συνήθως, αυτές οι προδιαγραφές δίνονται μέσω μιας Γλώσσας Ορισμού Διεπαφών (Interface Definition Language, IDL).

Κάθε IDL παρέχει απλώς ένα συντακτικό για την περιγραφή των παραμέτρων των διαδικασιών. Αυτό έχει τη δυνατότητα να είναι ανεξάρτητο από συγκεκριμένες γλώσσες προγραμματισμού για να διευκολύνει την επικοινωνία μεταξύ διαδικασιών γραμμένων σε διαφορετικές γλώσσες. Επιπλέον ενδέχεται να διαθέτει μόνο τις δυνατότητες που επιθυμούμε να υποστηρίζει το σύστημα RPC. Για παράδειγμα, μπορεί να μην υποστηρίζει παραμέτρους αναφοράς.

Αφού γράψουμε τις προδιαγραφές των διαδικασιών σε κάποια IDL, χρησιμοποιούμε μια γεννήτρια κορμών για να δημιουργήσουμε αυτόματα τον κορμό του πελάτη και του εξυπηρετητή. Η γεννήτρια συνδυάζει τον τυποποιημένο κώδικα για την υλοποίηση της επικοινωνίας μεταξύ των κορμών με τα κατάλληλα τμήματα κώδικα για τη διαχείριση των παραμέτρων κάθε διαδικασίας. Οι κορμοί που παράγονται από τη γεννήτρια συνδέονται με τις διαδικασίες πελάτη και εξυπηρετητή, αποκρύπτοντας το ότι ο πελάτης και εξυπηρετητής δεν συνδέονται μεταξύ τους. Να σημειώσουμε ότι η IDL δεν περιγράφει τη σημασιολογία των κλήσεων παρά μόνο τη σύνταξή τους, αφού αυτή επαρκεί για τη δημιουργία των κορμών.

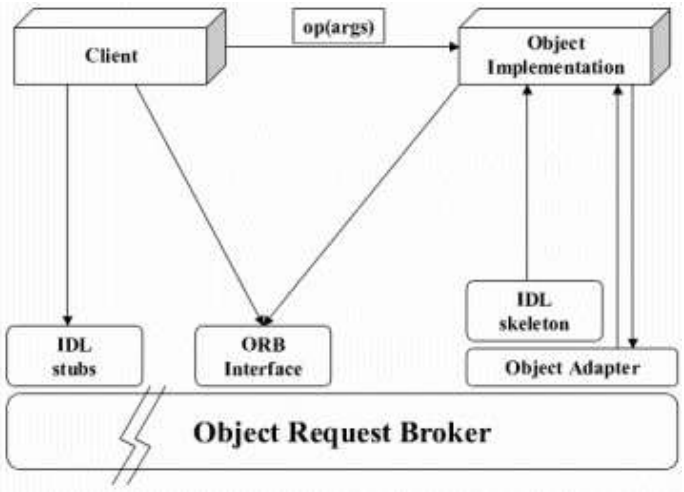
7.2. Σύστημα CORBA (Common Object Request Broker Architecture)

Η αρχιτεκτονική CORBA (*Common Object Request Broker Architecture*) επιτρέπει τη διασύνδεση προγραμμάτων και εξαρτημάτων ανεξαρτήτως κατασκευαστή, λειτουργικού συστήματος και γλώσσας προγραμματισμού.

Οι εφαρμογές CORBA αποτελούνται από αντικείμενα (*objects*). Ένα αντικείμενο μπορεί να αντιστοιχεί σε ένα αντικείμενο μιας γλώσσας προγραμματισμού, ή σε μία ολόκληρη εφαρμογή η οποία παρέχει συγκεκριμένες λειτουργίες. Με την CORBA δίνεται η δυνατότητα τα διαφορετικά κομμάτια λογισμικού(διαφορετικής ίσως πλατφόρμας) να αλληλεπιδρούν μεταξύ τους έτσι ώστε το σύστημα να παρέχεται σαν ένα ενιαίο σύνολο από λειτουργίες η υπηρεσίες. Για κάθε αντικείμενο γράφουμε μία περιγραφή της διεπαφής του σε IDL έτσι ώστε, λόγω της ανεξαρτησίας από την γλώσσα στην οποία είναι υλοποιημένο το αντικείμενο που προσφέρει η IDL, να επιτρέπεται η συνεργασία ετερογενών αντικειμένων. Συνήθως, κάθε γλώσσα προγραμματισμού που υλοποιεί CORBA διαθέτει και ειδικούς μεταγλωττιστές(IDL compiler) για την μεταγλώττιση IDL κώδικα σε κώδικα της αντίστοιχης γλώσσας που θα παραχθεί αυτόματα και θα μπορεί να γίνει compile με τον παραδοσιακό τρόπο στην γλώσσα αυτή.

Όταν ένα αντικείμενο καλεί ένα άλλο αντικείμενο, χρησιμοποιεί την περιγραφή του αντικειμένου χωρίς να γνωρίζει τίποτε για την υλοποίησή του. Η περιγραφή μεταγλωττίζεται σε αντιπροσώπους (*proxies*) του πραγματικού αντικειμένου. Ο αντιπρόσωπος του αντικειμένου-πελάτη ονομάζεται stub και ο αντιπρόσωπος του αντικειμένου-εξυπηρετητή ονομάζεται skeleton.

Όταν ένας αντικείμενο (ο πελάτης) θέλει να επικοινωνήσει με ένα άλλο αντικείμενο (τον εξυπηρετητή) καλεί την αντίστοιχη μέθοδο του stub(Σχήμα 18). Ένας μεσίτης (*Object Request Broker, ORB*) δρομολογεί την αίτηση στο κατάλληλο skeleton του εξυπηρετητή μέσω του αντίστοιχου Object Adapter, δουλειά του οποίου είναι ο υπολογισμός αναφορών προς το συγκεκριμένο αντικείμενο, οι πολιτικές δημιουργίας και πρόσβασης σε αυτό, κλπ.



Σχήμα 18: Αρχιτεκτονική CORBA

Κατά την κλήση τα εμπλεκόμενα αντικείμενα μπορεί να βρίσκονται σε διαφορετικά μηχανήματα. Στην περίπτωση αυτή, ο μεσίτης που εδρεύει στο ένα μηχάνημα επικοινωνεί με το μεσίτη που επικοινωνεί στο άλλο μηχάνημα χρησιμοποιώντας το κατάλληλο πρωτόκολλο (*IIOP, Internet Inter-ORB Protocol)*, χωρίς κανένα από τα εμπλεκόμενα αντικείμενα να καταλάβει ότι επικοινωνεί με αντικείμενο σε μία άλλη μηχανή.

Για την ανάπτυξη σε Java κώδικα που χρησιμοποιεί αντικείμενα CORBA απαιτούνται κάποια συγκεκριμένα βήματα. Το πρώτο βήμα είναι η ανάπτυξη του IDL κώδικα που περιγράφει αφηρημένα, ανεξαρτήτου πλατφόρμας, το αντικείμενο. Στην συνέχεια κάνουμε χρήστη του IDL Compiler(idlj tool) για τη μεταγλώττιση και μετατροπή του κώδικα IDL σε κατάλληλο Java κώδικα. Έπειτα δημιουργούμε την κλάση που υλοποιεί τις λειτουργίες του απομακρυσμένου αντικειμένου που θα βρίσκεται στο εξυπηρετητή και αφού αρχικοποιήσουμε το ORB(μηχανισμός επικοινωνίας), δημιουργούμε(instantiate) ένα αντικείμενο στον εξυπηρετητή, το οποίο το καταχωρούμε στην υπηρεσία ονοματοδοσίας (name service - μηχανισμός που παρέχεται στις υλοποιήσεις CORBA) του εξυπηρετητή με έναν συγκεκριμένο όνομα. Στο τέλος δημιουργούμε τον πελάτη στον οποίο αφού αρχικοποιήσουμε το ORB(μηχανισμός επικοινωνίας) κάνουμε κλήση του απομακρυσμένου αντικειμένου με το όνομα που είναι καταχωρημένο στην υπηρεσία ονοματοδοσίας του εξυπηρετητή και εν συνέχεια των μεθόδων-διαδικασιών που αυτό διαθέτει.

Σε γενικές γραμμές ο μηχανισμός για την προσπέλαση ενός κατανεμημένου αντικειμένου σε περιβάλλον CORBA είναι παρόμοιος με αυτόν του RMI ωστόσο υπάρχουν σημαντικές διαφορές μεταξύ των δυο τεχνολογιών, κυρίως στο ότι το RMI είναι μια καθαρή τεχνολογία Java ενώ η CORBA είναι μια τεχνολογία πολλών γλωσσών.

Παρακάτω παρατίθεται κώδικας για την κλήση των απομακρυσμένων μεθόδων getMessage() και getResult(int a, int b) του απομακρυσμένου αντικειμένου `DemoServiceImpl`:

Στην αρχή περιγράφουμε αφηρημένα το απομακρυσμένο αντικείμενο και τις μεθόδους που αυτό διαθέτει σε γλώσσα IDL και το αποθηκεύουμε σε αρχείο με κατάληξη idl.

```
module DemoServiceApp{

    interface DemoService{
        string getMessage();
        long getResult(in long numA, in long numB);
    };

};
```

Στην συνέχεια κάνουμε χρήστη του IDL Compiler(idlj tool) για τη μεταγλώττιση και μετατροπή του κώδικα IDL σε κατάλληλο Java κώδικα εκτελώντας τις κατάλληλες εντολές για τον εξυπηρετητή και για τον πελάτη ως εξής:

- Για τον Server: idlj –fall <όνομα αρχείου>.idl
- Για τον Client: idlj –fclient <όνομα αρχείου>.idl

Έπειτα δημιουργούμε την κλάση που υλοποιεί τις λειτουργίες του απομακρυσμένου αντικειμένου που θα βρίσκεται στο εξυπηρετητή:

```
public class DemoServiceImpl extends DemoServicePOA {

    //Οι μεθοδοι του απομακρυσμένου αντικειμένου προς κλήση.
    @Override
    public String getMessage() {
        return "Hello world!!!";
    }

    @Override
    public int getResult(int numA, int numB) {
        return numA + numB;
    }
}
```

Έπειτα δημιουργούμε την κλάση του εξυπηρετητή:

```
public class CORBADemoServer {

  public CORBADemoServer(String[] args) {
     try {
        // Αρχικοποίηση του ORB(Object Request Broker).
        ORB orb = ORB.init(args, null);

        // Έναρξη του διαχειριστή POA(Portable Object Adapter).
        POA rootpoa=POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
        rootpoa.the_POAManager().activate();

        // Κλήση της υπηρεσίας ονοματοδοσίας(Name Service) του ORB.
        org.omg.CORBA.Object ns = orb.resolve_initial_references("NameService");
        NamingContextExt ncExt = NamingContextExtHelper.narrow(ns);

        // Δημιουργία του απομακρυσμένου αντικειμένου.
        DemoServiceImpl dsi = new DemoServiceImpl();

        // Δημιουργία αναφοράς του απομακρυσμένου αντικειμένου.
        // Χρήση αντικειμένων που έχουν παραχθεί αυτόματα(IDL σε JAVA).
        org.omg.CORBA.Object ref = rootpoa.servant_to_reference(dsi);
        DemoService obj_ref = DemoServiceHelper.narrow(ref);

        // Δημιουργία ονόματος «DemoService»
        NameComponent obj_name[] = ncExt.to_name("DemoService");
        // Καταχώρηση αντικειμένου στην υπηρεσία ονοματοδοσίας.
        ncExt.rebind(obj_name, obj_ref);

        // Εν αναμονή αιτήσεων από πελάτες.
        System.out.println("Server started...");
        orb.run();

     } catch (Exception e) {
        e.printStackTrace(System.out);
     }
  }
  //Κύρια συνάρτηση. Δημιουργία εξυπηρετητή.
  public static void main(String[] args) {
     new CORBADemoServer(args);
  }
}
```

Στο τέλος δημιουργούμε την κλάση του πελάτη:

```
public class CORBADemoClient {

  public CORBADemoClient() {
     try {
        // Arguments για την αρχικοποίηση του ORB.
        String[] args = new String[]{"-ORBInitialPort", "5000",
                                     "-ORBInitialHost", "localhost"};
        // Αρχικοποίηση του ORB(Object Request Broker).
        ORB orb = ORB.init(args, null);

        // Κλήση της υπηρεσίας ονοματοδοσίας(Name Service) του ORB.
        org.omg.CORBA.Object ns=orb.resolve_initial_references("NameService");
        NamingContextExt ncExt = NamingContextExtHelper.narrow(ns);

        // Εύρεση του απομακρυσμένου αντικειμένου από την υπηρεσία ονοματοδοσίας
        org.omg.CORBA.Object obj = ncExt.resolve_str("DemoService");
        DemoService ds = DemoServiceHelper.narrow(obj);

        // Κλήση των απομακρυσμένων διαδικασιών του απομακρυσμένου αντικειμένου.
        String message = ds.getMessage();
        System.out.println("A: " + message);

        int result = ds.getResult(12, 13);
        System.out.println("B: " + result);

     } catch (Exception e) {
           System.out.println("ERROR : " + e);
           e.printStackTrace(System.out);
     }

  }

  //Κύρια συνάρτηση. Δημιουργία του πελάτη.
  public static void main(String[] args) {
     new CORBADemoClient();
  }
}
```

7.3. Ανάπτυξη CORBA συστήματος

Να υλοποιηθεί JAVA Client – Server μηχανισμός κατά τον οποίο οι χρήστες μπορούν να κάνουν κλήση απομακρυσμένων διεργασιών σύμφωνα με την αρχιτεκτονική CORBA. Ο Server θα διαθέτει δύο «απομακρυσμένα αντικείμενα» με μεθόδους σε καθένα από αυτά. Το πρώτο θα αφορά πράξεις(addition, subtraction, multiplication, division) δύο ακεραίων και το άλλο επεξεργασία δύο συμβολοσειρών(concat, equals, lengthDiff). Ο Client θα είναι μία διεπαφή η οποία θα αποτελείται από τουλάχιστον, 4 textfield, δύο για τα στοιχεία επικοινωνίας(διεύθυνση, πόρτα) του server και δύο για την εισαγωγή των δεδομένων(ακέραιοι η συμβολοσειρές), 1 textarea, για την εμφάνιση των αποτελεσμάτων και 2 buttons, ένα για την κλήση των μεθόδων του 1ου «απομακρυσμένου αντικείμενου» και ένα για την κλήση των μεθόδων του 2ου. Έτσι ο χρήστης συμπληρώνοντας τα στοιχεία του Server και τα πεδία με τα δεδομένα αναλόγως ποίο button θα πατήσει θα εκτελούνται μία-μία οι μέθοδοι του επιλεγμένου «αντικειμένου» και θα του επιστρέφονται τα δεδομένα τα οποία θα εμφανίζονται στο textarea όπως στις περιπτώσεις στο παρακάτω παράδειγμα:

a) //δεδομένα ‘5’ και ‘2’
5+2 ➔ 7
5–2 ➔ 3
5*2 ➔ 10
5/2 ➔ 2,5

b) //δεδομένα ‘geia’ και ‘geia’
concat ➔ geiageia
equals ➔ true
lengthDiff ➔ 0

Προσοχή στα παρακάτω

Θα πρέπει να δημιουργήσετε μια περιγραφή των «απομακρυσμένων αντικειμένων» με τις διεργασίες τους, με χρήση της γλώσσας IDL(Interface Description Language) σε ένα αρχείο με κατάληξη idl(<όνομα αρχείου>.idl).

Χρησιμοποιήστε τα ονόματα addition, subtraction, multiplication, division για τις μεθόδους του 1ου αντικειμένου και concat, equals, lengthDiff για τις μεθόδους του 2ου

Δημιουργήστε αυτόματα τις απαραίτητες Java κλάσεις χρησιμοποιώντας το εργαλείο μεταγλώττισης από IDL σε JAVA, ως εξής:

a) Για τον Server: idlj –fall <όνομα αρχείου>.idl
b) Για τον Client: idlj –fclient <όνομα αρχείου>.idl

Υλοποιήστε κατάλληλα τις κλάσεις των απομακρυσμένων αντικειμένων. Η concat θα παίρνει δύο string και θα τοποθετεί το δεύτερο στο τέλος του πρώτου (επιστρέφει string), η equals θα παίρνει δύο string και θα τα συγκρίνει για ισότητα (επιστρέφει boolean) και η lengthDiff θα υπολογίζει την διαφορά πλήθους των χαρακτήρων ανάμεσα στα δύο strings(επιστρέφει int).

Πριν τρέξετε τον CORBAServer θα πρέπει να εκκινήσετε τον Server της υπηρεσίας ονομάτων, στην ίδια <port> με το ORB του CORBAServer, πληκτρολογώντας στο command line> orbd –ORBInitialPort <port>.

8. Σχετική βιβλιογραφία

Σημειώσεις μαθήματος «Κατανεμημένα Συστήματα», Τμήμα Εφαρμοσμένης Πληροφορικής και Πολυμέσων, ΤΕΙ Κρήτης.

Κατανεμημένα συστήματα με Java, Ιωάννης Κ. Κάβουρας, Γιάννης Ζ. Μήλης, Κατερίνα Α. Ρουκουνάκη, Γιώργος Β. Ξυλωμένος, 3η Έκδοση.

Wikipedia, The Free Encyclopedia (http://en.wikipedia.org).

The Java Tutorials (http://docs.oracle.com/javase/tutorial/).

Java™ Platform, Standard Edition 6, API Specification (http://docs.oracle.com/javase/6/docs/api/).

The OMG(Object Management Group) specifications (http://www.omg.org/spec/index.htm).