ΟΝΟΜΑΣΙΑ ΙΔΡΥΜΑΤΟΣ: ΤΕΙ ΠΑΤΡΩΝ

ΣΧΟΛΗ: ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ

ΤΜΗΜΑ: ΕΠΙΧΕΙΡΗΜΑΤΙΚΟΥ ΣΧΕΔΙΑΣΜΟΥ ΚΑΙ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ:

Ανάπτυξη Εκπαιδευτικού Υλικού για τις Δομές Δεδομένων και Οργάνωση Αρχείων (στοίβες, ουρές, δέντρα) .

ΟΝΟΜΑΤΕΠΩΝΥΜΑ ΣΠΟΥΔΑΣΤΡΙΩΝ:

Μπίτζου Ελισάβετ

Παραπραστανίτη Όλγα

ΕΠΟΠΤΕΥΩΝ ΚΑΘΗΓΗΤΗΣ: Άρης Ν. Μπακάλης

ΠΑΤΡΑ, ΔΕΚΕΜΒΡΙΟΣ 2009

ΠΕΡΙΕΧΟΜΕΝΑ

1. Ιστορία της Java

Η γλώσσα της Java αρχικά αναπτύχθηκε από την Sun Microsystems υπό την αιγίδα των James Gosling και Bill Joy, σαν μέρος ενός ερευνητικού έργου ανάπτυξης λογισμικού για ηλεκτρονικές συσκευές καταναλωτικού επιπέδου (π.χ. video,τηλεόραση). Ο τρόπος αυτός χρησιμοποίησης της, την μετέτρεψε σε μία ιδανική γλώσσα για την διανομή εκτελέσιμων προγραμμάτων μέσω του WWW (Παγκόσμιου Ιστού) καθώς επίσης σε μία γενικού σκοπού γλώσσα προγραμματισμού για την ανάπτυξη προγραμμάτων τα οποία θα είναι εύκολα στην χρήση και θα μπορούν να μεταφέρονται σε διαφορετικά λειτουργικά συστήματα.

Αν και χρησιμοποιήθηκε από την Sun σε πολλές εφαρμογές (με το όνομα Οακ) με σκοπό την δημιουργία προϊόντων για την ηλεκτρονική αγορά, ενδιαφέρον προκάλεσε στο κοινό μόνο όταν συνδυάστηκε με το πρόγραμμα ανάγνωσης ιστοσελίδων (browser) της HotJava. Αυτό είχε ως αποτέλεσμα, η γλώσσα αυτή να συνδεθεί στενά με την ανάπτυξη μικρό - εφαρμογών στο διαδίκτυο και συγκεκριμένα στον Παγκόσμιο Ιστό (WWW). Η πραγματική όμως απογείωση της δημοτικότητας της Java ξεκίνησε όταν η Netscape ενσωμάτωσε την δυνατότητα της HotJava να τρέχει μικρό- εφαρμογές μέσα στο δικό της πρόγραμμα ανάγνωσης ιστοσελίδων (browser).

Το γεγονός, ότι τα τελευταία χρόνια η Java χρησιμοποιείται κυρίως για την ανάπτυξη μικρό - εφαρμογών δεν σημαίνει ότι η γλώσσα αυτή δεν είναι κατάλληλη για την δημιουργία ολοκληρωμένων εφαρμογών. Εξάλλου αυτό αποδεικνύεται από το γεγονός ότι τα περισσότερα εργαλεία της έχουν γραφτεί με την Java. Μάλιστα, σύμφωνα με την θεωρεία ανάπτυξης μεταγλωττιστών, μία γλώσσα έχει "ενηλικιωθεί" όταν ο μεταγλωττιστής της μπορεί να γραφτεί από την ίδια την γλώσσα. Συνεπώς η Java ως γλώσσα προγραμματισμού έχει ενηλικιωθεί.

Αυτό που θα πρέπει κανείς να θυμάται για την Java είναι ότι πρόκειται για μία καλά οργανωμένη και καλά εκφρασμένη γλώσσα προγραμματισμού που έχει δανειστεί πολλά θετικά χαρακτηριστικά από άλλες γλώσσες όπως τη C++, τη SmallTalk και τη Lisp, αφαίρεσε όμως όλα εκείνα τα στοιχεία αυτών των γλωσσών που ίσως οδηγούσαν σε σύγχυση τους χρήστες.

2. Εισαγωγή στη Java

Η γλώσσα προγραμματισμού Java σχεδιάστηκε για να δώσει λύσεις σε μία σειρά πρακτικών προβλημάτων που εμφανίζονταν σε όλες τις ήδη υπάρχουσες γλώσσες. Η γλώσσα στην οποία στηρίχτηκε αρκετά η προσπάθεια δημιουργίας αυτής της νέας γλώσσας ήταν η C++, μία αντικειμενοστραφής (object oriented) γλώσσα η οποία όμως πέρα από τα αρχικά προβλήματα, κυρίως σε πρακτικά ζητήματα, με τους υπάρχοντες μεταφραστές, παρουσίασε ουσιαστικά προβλήματα στην πορεία. Η λύση που προτάθηκε από τους μηχανικούς της Sun ήταν η δημιουργία μιας εντελώς νέας γλώσσας που θα είναι αντικειμενοστραφής και θα βασίζεται στη C++.

3. Ορισμός και χαρακτηριστικά της Java

Ο επίσημος ορισμός της Java, όπως αυτός δίνεται από τη Sun, έχει ως εξής:

Java: Μία απλή, αντικειμενοστραφής, έχουσα δικτυακή γνώση, μεταφράσιμη, εύρωστη, ασφαλής, ανεξάρτητη αρχιτεκτονικής, μεταφέρσιμη, υψηλής απόδοσης, πολυνηματική, δυναμική γλώσσα.

Για να γίνουν όμως κατανοητοί όλοι αυτοί οι χαρακτηρισμοί θα πρέπει να αναλυθεί ο κάθε ένας ξεχωριστά.

Απλή

Η Java προέκυψε από την C++ με την αφαίρεση ορισμένων χαρακτηριστικών. Συνεπώς, η Java είναι σίγουρα ευκολότερη από την C++. Αυτό σε συνδυασμό με το γεγονός ότι οι περισσότεροι προγραμματιστές πρώτα μαθαίνουν C++, οδηγεί στο συμπέρασμα ότι είναι πολύ πιο εύκολο για αυτούς να μάθουν την Java.

Η Java είναι πιο απλή γιατί περιλαμβάνει τρεις μόνο βασικούς τύπους δεδομένων: τους αριθμούς, τις λογικές μεταβλητές (Boolean) και τους πίνακες. Όλα τα υπόλοιπα στην Java αποτελούν μία κλάση. Για παράδειγμα, τα αλφαριθμητικά είναι αντικείμενα και όχι απλά ένας πίνακας χαρακτήρων.

Αναφέραμε πριν ότι η Java έχει αφαιρέσει ορισμένα χαρακτηριστικά από την C++, όπως είναι η δήλωση goto, η υπερφόρτωση των τελεστών (η οποία προκαλούσε σύγχυση στους προγραμματιστές της C++), οι δομές, τα ενωτικά όπως επίσης τις δηλώσεις #define και Typedef. Όλα αυτά τα χαρακτηριστικά ήταν απαραίτητα στην C++ προκειμένου να μπορέσει ο μεταγλωττιστής να μεταγλωττίσει σωστά τον ήδη υπάρχοντα κώδικα της C ο οποίος και σχετιζόταν με τα χαρακτηριστικά αυτά.

Το βασικό όμως στοιχείο της C++ το οποίο παραλήφθηκε από την Java είναι η άμεση διαχείριση της μνήμης, μέσα από την χρησιμοποίηση των δεικτών. Όπως ομολογούν οι περισσότεροι προγραμματιστές της C ή της C++, οι δείκτες αποτελούν την πηγή προβλημάτων τόσο κατά την διάρκεια γραφής του προγράμματος, όσο και κατά την μεταγλώττιση του. Οι δείκτες μπορούν

κατά λάθος να οριστούν από τον χρήστη να δείχνουν σε λάθος μεταβλητή, με αποτέλεσμα την μη σωστή λειτουργία του προγράμματος. Πολλές φορές μάλιστα αυτό μπορεί να οδηγήσει σε κατάρρευση του προγράμματος. Επίσης, οι δείκτες μπορούν να αποθηκεύουν κατανεμημένη μνήμη. Εάν η κατανεμημένη αυτή μνήμη δεν απελευθερωθεί το πρόγραμμα σιγά-σιγά θα δεσμεύει μνήμη μέχρις ότου να μην υπάρχει καθόλου διαθέσιμη μνήμη.

Έχουν δημιουργηθεί αρκετά προϊόντα, όπως το Bounds Checker, προκειμένου να βοηθήσουν τους προγραμματιστές να επιλύσουν τα προβλήματα αυτά που προκύπτουν από τους δείκτες. Η Java όμως, απλοποιεί το πρόβλημα αυτό με το να αφαιρεί την έννοια των δεικτών από την γλώσσα προγραμματισμού.

Βέβαια η Java δεν αφαιρεί μόνο στοιχεία από την C++, αλλά προσθέτει και νέα γιατί διαφορετικά δεν θα μιλούσαμε για μία νέα γλώσσα προγραμματισμού, αλλά για ένα φτωχό μεταγλωττιστή. Επομένως, ένα από τα βασικά χαρακτηριστικά που έχει προσθέσει η Java είναι η αυτόματη διαχείριση της μνήμης, γνωστό και ως "μονάδα συλλογής σκουπιδιών" (garbage collector). Ο προγραμματιστής δεν χρειάζεται να απελευθερώνει την μνήμη που κατανέμει- η εικονική μηχανή της Java (Virtual Machine ή VMJ) κάνει την δουλεία αυτή για λογαριασμό του προγραμματιστή.

Επίσης, υποστηρίζει τη δημιουργία πολυνηματικών προγραμμάτων. Ένα πρόγραμμα λέγεται πολυνηματικό, όταν σχεδιάζεται έτσι ώστε να μπορεί να εκτελεί περισσότερες από μία δουλείες ταυτόχρονα.

Αντικειμενοστραφής

Όσον αφορά στα αντικείμενα η Java ακολουθεί περισσότερο τα χαρακτηριστικά της γλώσσας SmallTalk παρά της C++. Όπως έχουμε ήδη αναφέρει εκτός από τους τύπους των δεδομένων, όλα τα υπόλοιπα στην Java αποτελούν αντικείμενα. Στην Java δεν υπάρχουν καθολικές συναρτήσεις: όλες οι συναρτήσεις καλούνται μέσα από ένα αντικείμενο. Σημαντικό επίσης είναι, ότι η Java δεν υιοθετεί τα χαρακτηριστικά της πολλαπλής κληρονομικότητας, καθώς κάτι τέτοιο θα έκανε πιο πολύπλοκα τα πράγματα.

Οι κλάσεις στην Java αποτελούνται από τις μεθόδους και τις μεταβλητές. Μέθοδοι λέγονται οι συναρτήσεις στις οποίες μπορεί να απευθύνεται το αντικείμενο μίας κλάσης. Μεταβλητές λέγονται τα δεδομένα που καθορίζουν την κατάσταση ενός αντικειμένου.

Διαμοιραζόμενη

Η Java επιτρέπει την δημιουργία διαμοιραζόμενων εφαρμογών από μία σειρά κλάσεων οι οποίες χρησιμοποιούνται σε δικτυακές εφαρμογές. Με την χρησιμοποίηση της URL κλάσης της Java, μία εφαρμογή μπορεί πολύ εύκολα να προσπελάσει έναν απομακρυσμένο διακομιστή.

Εύρωστη

Οι σχεδιαστές της Java προέβλεψαν ότι η γλώσσα αυτή επρόκειτο να χρησιμοποιηθεί για την επίλυση πολύπλοκων προγραμματιστικών προβλημάτων. Η δημιουργία ενός διαμοιραζόμενου, πολυνηματικού προγράμματος το οποίο μπορεί να τρέχει σε μία ποικιλία λειτουργικών

συστημάτων με διάφορους επεξεργαστές δεν είναι εύκολη δουλεία. Προκειμένου να βοηθήσουν στην επίλυση ενός τέτοιου είδους προβλήματος οι σχεδιαστές της Java δημιούργησαν μία πολύ δυνατή γλώσσα.

Η διαχείριση της μνήμης έχει απλουστευτεί στην Java με δύο τρόπους: Πρώτον, δεν χρησιμοποιούνται οι δείκτες και επομένως είναι αδύνατον ένα πρόγραμμα της Java να καταστρέψει δεδομένα ή να γράψει πάνω σε άλλα. Δεύτερον, η Java ακολουθεί την γλώσσα Lisp και SmallTalk οι οποίες επιτρέπουν την αυτόματη απελευθέρωση της μνήμης η οποία έχει καταμεριστεί αλλά δεν χρησιμοποιείται πια.

Ασφάλεια

Οι ισχυροί μηχανισμοί ασφάλειας της Java δρουν σε τέσσερα διαφορετικά επίπεδα της αρχιτεκτονικής του συστήματος. Κατ΄ αρχήν, η ίδια η γλώσσα Java σχεδιάστηκε να είναι ασφαλής και ο μεταγλωττιστής της Java διασφαλίζει ότι ο πηγαίος κώδικας δεν παραβιάζει τους κανόνες ασφάλειας. Κατά δεύτερον, όλος ο κώδικας σε μορφή bytecode που εκτελείται από το περιβάλλον χρόνου εκτέλεσης, ελέγχεται για να διασφαλιστεί ότι και αυτός επίσης υπακούει στους κανόνες ασφάλειας. Κατά τρίτον, η μονάδα φόρτωσης κλάσεων διασφαλίζει ότι οι κλάσεις δεν παραβιάζουν τους ισχύοντες περιορισμούς όταν φορτώνονται στο σύστημα.

Τέλος, η συγκεκριμένη ως προς το API ασφάλεια εμποδίζει τις μίνι - εφαρμογές να κάνουν καταστροφικά πράγματα στο σύστημα.

Ουδέτερη- Αρχιτεκτονική

Την δεκαετία του 1980 υπήρχε μεγάλη ποικιλία προσωπικών υπολογιστών. Μπορούσες να αγοράσεις υπολογιστή από την Apple, Commodone, Radio Shack, Atari και από την IBM. Συγκεκριμένα κάθε διαφορετικού τύπου Ηλεκτρονικός Υπολογιστής είχε το δικό του διαφορετικό λειτουργικό σύστημα. Επειδή, η ανάπτυξη λογισμικού απαιτούσε πολύ χρόνο, πολύ λίγο από το λογισμικό που αναπτυσσόταν για να χρησιμοποιηθεί από ένα Ηλεκτρονικό Υπολογιστή είχε εφαρμογή σε κάποιο άλλο διαφορετικό μηχάνημα.

Η λύση στο πρόβλημα αυτό ήρθε με την ανάπτυξη των Windows, των Macintosh της Apple και του Unix. Ωστόσο, η ανάπτυξη λογισμικού για την χρησιμοποίηση του ταυτόχρονα και από τα Windows NT, Unix και Macintosh εξακολουθεί να είναι μία δύσκολη εργασία.

Η Java έχει καταφέρει να λύσει το πρόβλημα της χρησιμοποίησης των εφαρμογών που έχουν αναπτυχθεί από διαφορετικά μηχανήματα. Το γεγονός ότι ο μεταγλωττιστής της Java δημιουργεί κώδικα εντολών γραμμένο σε byte ο οποίος μεταγλωττίζεται από τον μεταγλωττιστή της Java προσδίδει στην Java μία ουδέτερη αρχιτεκτονική. Έτσι, τα προγράμματα της Java μπορούν να εκτελούνται από οποιοδήποτε μηχάνημα ανεξαρτήτως λειτουργικού συστήματος.

Εύκολη στην μεταφορά

Βασικός στόχος της Java ήταν η δημιουργία φορητών εφαρμογών έτσι ώστε καθώς αναπτύσσονται νέες αρχιτεκτονικές (είτε λόγω των λειτουργικών συστημάτων, είτε λόγω των ίδιων των μηχανημάτων), το περιβάλλον της Java να μπορεί να λειτουργεί μέσα σε αυτές.

Στην Java όλοι οι τύποι δεδομένων (π.χ. ακέραιοι, κινητής υποδιαστολής και κινητής υποδιαστολής διπλής ακριβείας) έχουν προκαθορισμένο μέγεθος ανεξάρτητα από το μηχάνημα ή το λειτουργικό σύστημα στο οποίο το πρόγραμμα τρέχει. Αυτό βέβαια έρχεται σε αντίθεση με την C++, όπου το μέγεθος των παραπάνω τύπων εξαρτάται από τον μεταγλωττιστή.

Υψηλή απόδοση

Μία εφαρμογή που αναπτύχθηκε με την Java σίγουρα δεν έχει την αποδοτικότητα που θα είχε αν είχε γραφτεί σε C++. Ωστόσο, για τις περισσότερες εφαρμογές που περιλαμβάνουν γραφικά τα οποία συναντάμε κυρίως στο διαδίκτυο, η απόδοση της Java είναι περισσότερο από αρκετή. Για ορισμένες μάλιστα εφαρμογές δεν υπάρχει ορατή διαφορά στην απόδοση ανάμεσα στην Java και την C++.

Πάντως, ένα είναι σίγουρο ότι η Java είναι αρκετά γρήγορη και μπορεί να επιτρέψει στους προγραμματιστές να κάνουν πράγματα που δεν θα μπορούσαν με την C++.

Πολυνηματική

Οι σύγχρονες εφαρμογές χρειάζεται να εκτελούν περισσότερες από μία λειτουργίες κάθε φορά. Η Java ενισχύει την ανάπτυξη τέτοιων εφαρμογών μέσω των νημάτων. Τα νήματα είναι μία ροή εκτέλεσης μέσα σε μία εφαρμογή. Ένα νήμα μπορεί να εκτελείται ανεξάρτητα σε μία εφαρμογή ή πολλά νήματα μπορούν να εκτελούνται ταυτόχρονα.

Η χρησιμοποίηση συγχρονισμένων νημάτων είναι εξαιρετικά χρήσιμη στην δημιουργία διαμοιραζόμενων, δικτυακών εφαρμογών. Ένα παράδειγμα τέτοιας εφαρμογής είναι η επικοινωνία με ένα απομακρυσμένο διακομιστή μέσω ενός νήματος, ενώ με ένα άλλο νήμα πραγματοποιείται η αλληλεπίδραση με τον χρήστη.

Όταν ζητάμε από τη Java να μεταγλωττίσει το πρόγραμμα μας αυτή δεν παράγει ένα πρόγραμμα όπως αυτά που έχουμε συνηθίσει να βλέπουμε. Αντί να δημιουργήσει ένα εκτελέσιμο αρχείο γεμάτο από εντολές μηχανής, ο μεταγλωττιστής της Java παράγει στην έξοδο αυτό που είναι γνωστό ως κώδικας byte Java (Java byte codes). Ο κώδικας byte Java είναι εντολές γραμμένες για κάποια εικονική μηχανή Java (virtual machine) που δεν υπάρχει πραγματικά. Για να εκτελέσουμε το πρόγραμμά μας, πρέπει να έχουμε ένα ερμηνευτή κώδικα byte java, πράγμα που σημαίνει ότι στην πραγματικότητα εξομοιώνουμε την εικονική μηχανή Java (JVM) σε όποιον υπολογιστή και όποιο λειτουργικό σύστημα χρησιμοποιούμε. Κανονικά δεν θα το αντιλαμβανόμαστε αυτό, αφού η Java το κάνει αυτόματα όταν εκτελούμε την εφαρμογή μας μέσα στο Αλληλεπιδραστικό Περιβάλλον Ανάπτυξης (Interactive Development Environment IDE) της Java.

4.Εγκατάσταση Java

Η έκδοση της Java που χρησιμοποιήσαμε στην πτυχιακή μας εργασία είναι η BlueJ version 2.2.1 και διατίθεται δωρεάν στο Internet στο ακόλουθο site:

http://www.bluej.org/download/download.html

Η διαδικασία εγκατάστασης περιλαμβάνει τα ακόλουθα βήματα:

1.Σύνδεση στην τοποθεσία

http://java.sun.com/javase/downloads/index_jdk5.jsp.

2. Εκεί στην επιλογή JDK 5.0 Update 16 πατάμε το πλήκτρο Download για να κατεβάσουμε και να εγκαταστήσουμε τη Java στον Η/Υ μας.

3.Έπειτα επιλέγουμε τι λειτουργικό σύστημα χρησιμοποιούμε, επιλέγουμε multilanguge και κλικάρουμε την επιλογή "I agree to the Java Development Kit 5.0 License Agreement".

4. Τέλος επιλέγουμε την επιλογή "Windows Offline Installation" και κλικαρουμε πανω στο

jdk-1_5_0_16-windows-i586-p.exe για να αρχίσει το download.

Με αυτό τον τρόπο κατεβάζουμε και εγκαθιστούμε στον Η/Υ μας την πλατφόρμα της Java η οποία είναι απαραίτητη για να λειτουργήσει το BlueJ. Η συγκεκριμένη πλατφόρμα που αναφέρουμε είναι η έκδοση JDK 1.5. Μπορούμε να εγκαταστήσουμε και οποιαδήποτε άλλη πλατφόρμα αντί αυτής είτε μεταγενέστερης είτε προηγούμενη έκδοσης.

Για να κατεβάσουμε το BlueJ κάνουμε τα εξής:

1. Σύνδεση στην τοποθεσία http://www.bluej.org/download/download.html

2. Στην επιλογή BlueJ version 2.2.1 for Windows (3.6Mb) πατάμε bluejsetup-221.exe για να κατεβάσουμε και να εγκαταστήσουμε το BlueJ στον Η/Υ μας.

Παρατήρηση:

Στη παραπάνω τοποθεσία (http://www.bluej.org/download/download.html) η έκδοση του BlueJ ανανεώνεται , οπότε εάν δεν υπαρχει η συγκεκριμένη (2.2.1)μπορούμε να κατεβάσουμε την ποιο καινούργια.

5.Διαδικασία γραφής ενός προγράμματος στη γλώσσα Java

Για να γράψουμε ένα πρόγραμμα στη γλώσσα Java θα πρέπει αρχικά να έχουμε εγκαταστήσει ένα ολοκληρωμένο περιβάλλον συγγραφής, μετάφρασης και εκτέλεσης προγραμμάτων σε Java. Η διαδικασία αυτή αναφέρεται διεξοδικά στην παραπάνω. Στη συνέχεια πρέπει να ανοίξουμε το περιβάλλον αυτό για να γράψουμε και να εκτελέσουμε προγράμματα σε Java. Η διαδικασία αυτή περιγράφεται στα ακόλουθα βήματα (σε περιβάλλον Windows XP):

Πηγαίνουμε Έναρξη -> Όλα τα προγράμματα ->BlueJ -> Κάνουμε κλικ στο εικονίδιο με όνομα BlueJ: 

. Εναλλακτικά μπορούμε να ανοίξουμε τη συντόμευση για το BlueJ που μπορεί να υπάρχει στην επιφάνεια εργασίας

Ανοίγει το πρόγραμμα Blue J και εμφανίζεται μια κενή οθόνη. Αν έχουμε ήδη δημιουργήσει κάποιο project σε Java τότε ανοίγει αυτόματα το τελευταίο project με το οποίο εργαζόμασταν.

Για να δημιουργήσουμε ένα νέο πρόγραμμα με το Blue J κάνουμε τα εξής:

1. Πηγαίνουμε στο μενού Project και επιλέγουμε new project. Το πρόγραμμα ζητάει να ορίσουμε το όνομα του νέου project και να επιλέξουμε το φάκελο στον οποίο θα αποθηκευτεί. Εναλλακτικά μπορούμε να δημιουργήσουμε ένα νέο φάκελο πατώντας στο παράθυρο διαλόγου New Project που είναι ανοιχτό μπροστά μας το εικονίδιο Create New Folder. Στο τέλος πατάμε το πλήκτρο create για να ολοκληρώσουμε τη δημιουργία του νέου project. Θα πρέπει να επισημάνουμε ότι ένα νέο project αποτελεί στην πραγματικότητα ένα νέο φάκελο στο οποίο θα τοποθετούνται αυτόματα τα αρχεία που θα δημιουργήσουμε στη συνέχεια.

2. Στη συνέχεια πατάμε το πλήκτρο New Class για να δημιουργήσουμε ένα νέο αρχείο-πρόγραμμα Java μέσα στο τρέχον project. Δίνουμε ένα όνομα για το νέο πρόγραμμα και πατάμε OK. Θα πρέπει να τονίσουμε στο σημείο αυτό ότι σε κάθε αρχείο που δημιουργούμε προστίθεται αυτόματα στο τέλος του η κατάληξη java.

3. Στη συνέχεια ανοίγουμε το αρχείο που δημιουργήσαμε είτε με διπλό κλικ επάνω του είτε με δεξί κλικ και Άνοιγμα οπότε ανοίγει το περιβάλλον του κειμενογράφου του BlueJ στο οποίο πληκτρολογούμε τον κώδικα που θέλουμε να περιλαμβάνει το νέο πρόγραμμα, αφού πρώτα διαγράψουμε το περιεχόμενο του κειμενογράφου. Στη συνέχεια παρουσιάζεται το αρχείο HelloWorld το οποίο ανοίγουμε με τον τρόπο που προαναφέραμε και πληκτρολογούμε τον ακόλουθο κώδικα μέσα σε αυτό:

```
public class HelloWorld
        {
          public static void main (String [] args)
         {
           System.out.println ("Hello World!");
         }
        }
```

4. Όταν ολοκληρώσουμε την πληκτρολόγηση του πηγαίου προγράμματος το μεταφράζουμε σε γλώσσα μηχανής πατώντας το εικονίδιο-εργαλείο Compile. Το εργαλείο αυτό ελέγχει τον κώδικα για συντακτικά λάθη τα οποία εντοπίζει και παρουσιάζει στο κάτω μέρος της οθόνης μαζί με τον αντίστοιχο αριθμό γραμμής στην οποία έχουν σημειωθεί. Στη περίπτωση που ο compiler δεν εντοπίσει λάθη τότε θα εμφανιστεί το μήνυμα "class compiled - no syntax errors στο κάτω μέρος του προγράμματος.

Επισημαίνουμε για διευκόλυνση των χρηστών του BlueJ ότι με το εικονίδιο Compile εκτός από συντακτικό έλεγχο κάνουμε και αποθήκευση στο περιεχόμενο του αρχείου. Επίσης θα πρέπει να σημειώσουμε ότι σε κάθε μεταφρασμένο αρχείο προστίθεται αυτόματα στο τέλος του η κατάληξη class. Τέλος πατάμε το εικονίδιο Close για να κλείσουμε το περιβάλλον του κειμενογράφου του BlueJ.

5. Για να εκτελέσουμε το πρόγραμμα πατάμε δεξί κλικ επάνω του (π.χ. δεξί κλικ στο εικονίδιο του αρχείου Helloword) και επιλέγουμε void main (String[] args). Αν το πρόγραμμα περιλαμβάνει και άλλες συναρτήσεις εκτός από το main (κύρια συνάρτηση) τότε στο παράθυρο αυτό εμφανίζονται και αυτές οι συναρτήσεις τις οποίες επίσης μπορούμε να εκτελέσουμε κάνοντας κλικ επάνω του.

Στο σημείο μπορούμε να εισάγουμε προαιρετικά ορίσματα-δεδομένα στο πρόγραμμα μας. Τα δεδομένα αυτά πρέπει να τοποθετούνται πάντα μέσα σε διπλά εισαγωγικά διότι η Java αντιμετωπίζει όλα τα δεδομένα που εισάγουμε σε αυτό το σημείο ως αλφαριθμητικά (strings). Για την εκτέλεση του προγράμματος HelloWorld δεν απαιτείται η εισαγωγή δεδομένων. Στο τέλος πατάμε OK και ανοίγει το παράθυρο εκτέλεσης όπου φαίνονται τα αποτελέσματα του προγράμματος.

6. Δομή προγράμματος

Ένα πρόγραμμα σε java έχει την ακόλουθη μορφή:

Κώδικας	Σχόλια-Παρατηρήσεις
public class όνομα	το όνομα της κλάσης πρέπει να είναι ίδιο με το όνομα του αρχε
{	αρχή κλάσης (χώρος εργασίας για συγγραφή συναρτήσεων)
public static void main(String [] args)	δήλωση κύριας συνάρτησης (κύριου προγράμματος)
{	αρχή κύριας συνάρτησης
δήλωση μεταβλητών;	
εντολή 1;	
εντολή 2;	
εντολή 2;	
…….	
Εντολή v;	
}	τέλος κύριας συνάρτησης
}	τέλος κλάσης

Το πρόγραμμα Hello World που αναφέραμε στην προηγούμενη ενότητα είναι ίσως το πιο απλό πρόγραμμα που μπορεί κάποιος να γράψει σε Java. Παρόλα αυτά μας ενδιαφέρει για να δώσουμε μια μικρή περιγραφή για τη βασική δομή των προγραμμάτων σε Java. Ας το εξετάσουμε γραμμή προς γραμμή:

```
public class HelloWorld

{

public static void main (String [] args)

{

System.out.println ("Hello World!");

}

}
```

Η αρχική δήλωση class προσδιορίζει το όνομα του προγράμματος, στη συγκεκριμένη περίπτωση Hello World. Ο compiler παίρνει το όνομα από τη δήλωση class HelloWorld στο αρχείο πηγαίου κώδικα. Το Hello World class περιλαμβάνει μία μέθοδο, τη main. Όπως και στη C++, η μέθοδος main μας δείχνει από που ξεκινά να εκτελείται μία εφαρμογή. Η μέθοδος δηλώνεται δημόσια (public) δηλαδή μπορεί να καλείται παντού. Δηλώνεται στατική (static) δηλαδή όλα τα στιγμιότυπα της κλάσης μοιράζονται την ίδια μέθοδο. Δηλώνεται κενή (void) που σημαίνει ότι αυτή η μέθοδος δεν επιστρέφει τιμή-αποτέλεσμα όπως και στη C++.

Όταν καλείται η μέθοδος main τυπώνει το Hello World! στην οθόνη. Αυτό επιτυγχάνεται με τη μέθοδο System.out.println. Για να είμαστε πιο ακριβείς, αυτό επιτυγχάνεται καλώντας τη συνάρτηση println() του πεδίου out που ανήκει στην κλάση System. Η συνάρτηση println() χρησιμοποιείται γενικά για να τυπώνει μηνύματα, σχόλια και αποτελέσματα στην οθόνη του Η/Υ και είναι αντίστοιχη με την εντολή cout στη C++.

7. Αλφάβητο γλώσσας

Κάθε γλώσσα προγραμματισμού έχει το δικό της αλφάβητο από το οποίο σχηματίζει τις λέξεις ή τα σύμβολα που αποτελούν το λεξιλόγιο της. **Το αλφάβητο της Java περιλαμβάνει τους ακόλουθους χαρακτήρες:**

οριζόντιος στηλοθέτης (αναπαριστάνεται με το σύμβολο \t)

νέα γραμμή (αναπαριστάνεται με το σύμβολο \n)

γράμματα αγγλικής αλφαβήτου (κεφαλαία και πεζά)

αριθμοί

σύμβολα πράξεων + - * / %

σύμβολα στίξης ( ) & { } [ ] , . κ.λ.π.

8. Λεξιλόγιο γλώσσας

Το λεξιλόγιο της Java περιλαμβάνει:

- δεσμευμένες λέξεις (reserved words)

Οι δεσμευμένες λέξεις έχουν αυστηρά καθορισμένη έννοια και τρόπο χρήσης

και χρησιμοποιούνται σε συγκεκριμένες θέσεις ενός προγράμματος. Μερικές από

τις πιο χαρακτηριστικές δεσμευμένες λέξεις της Java είναι int, float, class, do, if, for, κ.λπ.

- τελεστές (operators) οι οποίοι περιγράφονται σε επόμενη ενότητα
- αναγνωριστές (identifiers)

Στην κατηγορία των αναγνωριστών (identifiers) ανήκουν λέξεις που κατασκευάζει ο προγραμματιστής. Τέτοιες λέξεις δίνει ο προγραμματιστής σε μεταβλητές, σταθερές. συναρτήσεις και δικούς του τύπους δεδομένων.

9. Αριθμητικοί τύποι δεδομένων

Κάθε τύπος δεδομένων έχει μία κύρια περιοχή τιμών. Ο μεταγλωττιστής δεσμεύει χώρο μνήμης για να αποθηκεύσει κάθε μεταβλητή ή σταθερά σύμφωνα με τον τύπο δεδομένων τους. Η Java παρέχει αρκετούς βασικούς (ενσωματωμένους) τύπους δεδομένων για αριθμητικές τιμές, χαρακτήρες Boolean τιμές κ.λπ.

Οι πιο βασικοί αριθμητικοί τύποι στη Java είναι οι ακόλουθοι:

Όνομα	Εύρος τιμών	Μέγεθος Μνήμης
byte	-2^7 (- 128) έως 2^7 ☐ 1 (127)	8-bit signed
short	-2^{15} (- 32768) έως 2^{15} ☐ 1 (32767)	16-bit signed
int	-2^{31} (-2147483648) έως 2^{31} ☐ 1 (2147483647)	32-bit signed
long	-2^{63} έως 2^{63} - 1 (π.χ.. ☐ 9223372036854775808 μέχρι 9223372036854775807)	64-bit signed
float	- 3.4E38 έως 3.4E38	32-bit
double	- 1.7E308 έως 1.7E308	64-bit

9.1 Προσαρμογή τύπων

Γενικά η Java μπορεί να κάνει αυτόματα μετατροπή από έναν αριθμητικό τύπο σε ένα μεγαλύτερο τύπο χρησιμοποιώντας μια προσαρμογή. Μια προσαρμογή (cast) είναι μια ρητή μετατροπή μιας τιμής, από τον τρέχοντα τύπο της σε έναν άλλο. Η προσαρμογή εμφανίζεται με τον επιθυμητό τύπο μέσα σε παρενθέσεις πριν από την τιμή που πρόκειται να μετατραπεί, όπως εδώ:

```
float fvalue;

int ivalue = (int) fvalue; //μετατροπή τύπου float σε int

double grades;

float grade = (float) grades; //μετατροπή τύπου double σε float
```

10. Δήλωση μεταβλητών

Οι μεταβλητές χρησιμοποιούνται για να αναπαραστήσουν πολλούς διαφορετικούς τύπους δεδομένων. Για να χρησιμοποιήσουμε μια μεταβλητή, τη δηλώνουμε με το όνομα της καθώς και τον τύπο δεδομένων που αναπαριστά. Αυτό ονομάζεται δήλωση μεταβλητής. Δηλώνοντας μια μεταβλητή δίνουμε τη δυνατότητα στο μεταγλωττιστή της γλώσσας να δεσμεύει την απαιτούμενη μνήμη, που ορίζει ο τύπος της, για τη μεταβλητή αυτή. Η σύνταξη για δήλωση μιας μεταβλητής είναι η ακόλουθη:

```
τύπος δεδομένων όνομα μεταβλητής;
```

Παραδείγματα δηλώσεων μεταβλητών

int x; // Δήλωση της x ως ακέραια;

double radius; // Δήλωση radius ως πραγματική μεταβλητή;

char a; // Δήλωση της a ως μεταβλητή χαρακτήρας;

String name="Nikos";

10.1 Δήλωση και αρχικοποίηση μεταβλητών σε ένα βήμα

Οι μεταβλητές συχνά έχουν αρχικές τιμές. Μπορούμε να δηλώσουμε μια μεταβλητή και να την αρχικοποιήσουμε σε ένα βήμα. Για παράδειγμα οι επόμενες εντολές είναι δήλωση και αρχικοποίηση διάφορων μεταβλητών:

int x=0; // Δήλωση της x ως ακέραια και αρχικοποίηση της με 0

double radius=1.0; // Δήλωση radius ως πραγματική μεταβλητή και αρχικοποίηση της με 1.0

char a='α'; //Δήλωση της a ως μεταβλητή χαρακτήρας και αρχικοποίηση της με το χαρακτήρα α

String name="Nikos"; //Δήλωση της a ως αλφαριθμητική μεταβλητή αρχικοποίηση της με τη συμβολοσειρά Nikos

11. Σταθερές

Η τιμή μιας μεταβλητής μπορεί να αλλάξει κατά τη διάρκεια εκτέλεσης του προγράμματος. Μια σταθερά αναπαριστά μόνιμα δεδομένα τα οποία ποτέ

δεν αλλάζουν. Π.χ. το π είναι μια σταθερά που αν τη χρησιμοποιούμε συχνά δεν θέλουμε να πληκτρολογούμε συνεχώς την τιμή 3.14159. Αντίθετα μπορούμε να καθορίσουμε μια σταθερά για το Π. Η σύνταξη για να καθορίσουμε μια σταθερά είναι η εξής:

```
final τύπος δεδομένων όνομα σταθεράς = τιμή;
```

Η λέξη final είναι λέξη κλειδί της Java το οποίο σημαίνει ότι η σταθερά δεν μπορεί να αλλάξει.

Παράδειγμα

```
double radius, area;

final double pi = 3.14159;
```

11.1 Αριθμητικές σταθερές

Μια αριθμητική σταθερά είναι ένας βασικός τύπος τιμής ο οποίος εμφανίζεται κατευθείαν στο πρόγραμμα. Για παράδειγμα 34, 1000000 και 5.0 είναι αριθμητικές σταθερές στις ακόλουθες εντολές:

```
int i = 34

long l = 1000000;

double d = 5.0;
```

Οι αριθμητικές σταθερές πραγματικού τύπου είναι γραμμένα με ένα δεκαδικό σημείο. Από λάθος μια σταθερά πραγματικού τύπου χρησιμοποιείται ως τύπου διπλής ακρίβειας. Για παράδειγμα 5.0 θεωρείται μια διπλής ακρίβειας τιμή όχι μια πραγματική τιμή. Μπορούμε να φτιάξουμε ένα αριθμητικό ή ένα πραγματικό αριθμό προσθέτοντας το γράμμα f, F, d ή D. Για παράδειγμα μπορούμε να χρησιμοποιήσουμε 100.2f ή 100.2F για ένα αριθμητικό αριθμό και 100.2d ή 100.2D για ένα πραγματικό αριθμό. Μια αριθμητική σταθερά μπορεί επίσης να γραφτεί με επιστημονική παράσταση. Για παράδειγμα 1.23456e + 2 που είναι ισοδύναμο του 1234565x 10^2 = 123.456.

12. Τελεστές

12.1 Σχεσιακοί τελεστές

Οι σχεσιακοί τελεστές χρησιμοποιούνται για να δημιουργήσουν λογικές συνθήκες και είναι οι ακόλουθοι:

Σχεσιακοί Τελεστές
< (μικρότερο)
<=(μικρότερο ή ίσο)
> (μεγαλύτερο)
>=(μεγαλύτερο ή ίσο)
== (ισότητα)
!= (διάφορο)

Παραδείγματα
if (x==3) ……
while (y>=10) ….
if (z!=0) ….
if (a>0) ….
if (b<20) ….

Σημείωση

Πρέπει να τονίσουμε τη διαφορά ανάμεσα στην εντολή καταχώρισης η οποία συμβολίζεται με το = και τον τελεστή της ισότητας ο οποίος συμβολίζεται με ==.

12.2 Λογικοί τελεστές

Οι λογικοί τελεστές χρησιμοποιούνται προκειμένου να δημιουργήσουμε σύνθετες συνθήκες από απλούστερες συνθήκες. Η τιμή μιας συνθήκης απλής ή σύνθετης είναι είτε true (αληθής) είτε false (ψευδής). Στη γλώσσα Java οι λογικοί τελεστές είναι οι ακόλουθοι είναι οι ακόλουθοι:

Λογικοί Τελεστές
&& (AND) ή &(AND)
|| (OR) ή >(OR)
! (NOT)

12.2.1 Λογικός τελεστής && (and) ή & (and)

Ο λογικός τελεστής && (ή &) δίνει την τιμή true όταν όλες οι συνθήκες που ελέγχει είναι ταυτόχρονα αληθείς. Αν έστω και μια από τις συνθήκες που ελέγχει είναι ψευδής τότε δίνει την τιμή false. Η σύνταξη του είναι η ακόλουθη:

(συνθήκη1 && συνθήκη2 && συνθήκη3..)

ή

(συνθήκη1 & συνθήκη2 & συνθήκη3..)

Ο πίνακας αλήθειας του λογικού τελεστή && για 2 συνθήκες φαίνεται στον ακόλουθο πίνακα:

Πίνακας αλήθειας τελεστή AND (&& ή &)		
Συνθήκη 1	Συνθήκη 2	Αποτέλεσμα
Α	Α	Α
Α	Ψ	Ψ
Ψ	Α	Ψ
Ψ	Ψ	Ψ

12.2.2 Λογικός τελεστής|| (or)

Ο λογικός τελεστής || δίνει την τιμή true όταν έστω και μια από τις συνθήκες συνθήκη που ελέγχει είναι αληθής αλλιώς δίνει την τιμή false. Η σύνταξη του είναι η ακόλουθη:

(συνθήκη1 || συνθήκη2 || συνθήκη3..)

ή

(συνθήκη1 | συνθήκη2 | συνθήκη3..)

Ο πίνακας αλήθειας του λογικού τελεστή || για 2 συνθήκες φαίνεται στον ακόλουθο πίνακα:

Πίνακας Αλήθειας τελεστή OR (|| ή >)		
Συνθήκη 1	Συνθήκη 2	Αποτέλεσμα
Α	Α	Α
Α	Ψ	Α
Ψ	Α	Α
Ψ	Ψ	Ψ

12.2.3 Λογικός τελεστής ! (not)

Ο λογικός τελεστής ! εφαρμόζεται σε μια μόνο συνθήκη και αντιστρέφει την τιμή της δηλαδή αν η συνθήκη είναι αληθής τότε δίνει την τιμή ψευδής (false) ενώ αν είναι ψευδής τότε δίνει την τιμή αληθής (true). Η σύνταξη του είναι η ακόλουθη:

!(συνθήκη)

Ο πίνακας αλήθειας του λογικού τελεστή ! φαίνεται στον ακόλουθο πίνακα:

Πίνακας Αλήθειας τελεστή NOT (!)	
Συνθήκη	Αποτέλεσμα
Α	Ψ
Ψ	Α

Σημείωση:

Οι λογικοί τελεστές && και || μπορούν γενικά να εφαρμοστούν σε 2 ή περισσότερες συνθήκες. Η προτεραιότητα των λογικών τελεστών είναι κατά σειρά:

! (not)

&& (and)

|| (or).

Συνθήκη	Χαρακτηρισμός
if (x!=0)	Απλή συνθήκη. Έχει τιμή true αν το x είναι διάφορο του 0.
if (x>=0 && x<5)	Σύνθετη συνθήκη. Έχει τιμή true μόνο αν ικανοποιούνται και οι 2 συνθήκες.
if (x==9 || x==10)	Σύνθετη συνθήκη. Έχει τιμή true αν ικανοποιείται είτε η μια είτε η άλλη συνθήκη.

if !(x<0)	Απλή συνθήκη. Έχει τιμή true αν το x είναι μεγαλύτερο ή ίσο του 0.
while (x>=0 && x<=10) || (x>=20 && x<=30)	Εντολή επανάληψης που ελέγχει μια σύνθετη συνθήκη. Η επανάληψη εκτελείται αν η σύνθετη συνθήκη έχει την τιμή true ενώ σταματά η εκτέλεση της αν η σύνθετη συνθήκη έχει την τιμή false. Η σύνθετη συνθήκη έχει την τιμή true αν έστω και μια από τις 2 συνθήκες στην παρένθεση είναι true.
for (i=0; x!=-1; i++)	Εντολή επανάληψης που ελέγχει μια απλή συνθήκη. Η επανάληψη εκτελείται αν η συνθήκη είναι αληθής.

12.3 Αριθμητικοί τελεστές

Οι αριθμητικοί τελεστές χρησιμοποιούνται για την εκτέλεση πράξεων. Στη γλώσσα Java είναι οι ακόλουθοι:

Αριθμητικοί Τελεστές
+ (άθροισμα)
- (διαφορά)
* (γινόμενο)
/ (πηλίκο)
% (υπόλοιπο)

13 Εντολές Εισόδου-Εξόδου

13.1 Εντολή Εισόδου

Στη Java μπορούμε να εισάγουμε δεδομένα με 2 διαφορετικούς τρόπους: είτε από τη γραμμή εντολών (command line) είτε από το πληκτρολόγιο. Ακολούθως περιγράφουμε διεξοδικά αυτούς τους 2 τρόπους.

Α Τρόπος: Εισαγωγή Δεδομένων από Γραμμή εντολών

Όλα τα δεδομένα που εισάγουμε στη γραμμή εντολών κατά την εκτέλεση ενός προγράμματος Java θεωρούνται συμβολοσειρές (strings). Για να τα μετατρέψουμε στον επιθυμητό τύπο δεδομένων χρησιμοποιούμε τις ακόλουθες συναρτήσεις μετατροπής οι οποίες προσφέρονται από την ίδια τη γλώσσα:

Συνάρτηση	Λειτουργία
Integer.parseInt(String)	μετατροπή συμβολοσειράς αριθμητικών ψηφίων στο αντίστοιχο ακέραιο
Float.parseFloat(String)	μετατροπή συμβολοσειράς αριθμητικών ψηφίων στο αντίστοιχο πραγματικό απλής ακρίβειας
Double.parseDouble(String)	μετατροπή συμβολοσειράς αριθμητικών ψηφίων στο αντίστοιχο πραγματικό διπλής ακρίβειας

Δεν υπάρχει έτοιμη συνάρτηση για εισαγωγή χαρακτήρα. Αυτό μπορεί να γίνει εισάγοντας το χαρακτήρα ως ένα αλφαριθμητικό με μέγεθος 1 χαρακτήρα και μετά παίρνοντας το χαρακτήρα αυτό μέσα από το αλφαριθμητικό με τη συνάρτηση charAt(θέση). Για παράδειγμα γράφοντας: char operator=args[0].charAt(0); σημαίνει ότι από το 1ο όρισμα που δίνουμε στη γραμμή εντολών παίρνουμε τον αρχικό χαρακτήρα του δηλαδή αυτόν που βρίσκεται στη θέση 0 και τον καταχωρούμε στη μεταβλητή operator τύπου char.

Β Τρόπος: Εισαγωγή Δεδομένων από Πληκτρολόγιο

Η εντολή (συνάρτηση) εισόδου για διάβασμα δεδομένων από το πληκτρολόγιο είναι η readLine() η οποία διαβάζει επίσης μια συμβολοσειρά (string) από το πληκτρολόγιο. Υλοποιείται με τον ακόλουθο κώδικα:

```
public class eisagwgi_dedomenwn
{
public static String readstring()
{
    BufferedReaderbr=newBufferedReader(newInputStreamReader
    (System.in), 1);
    String st=" ";
    try
    {
```

```
        st=br.readLine();
    }
    catch (IOException ex)
    {
        System.out.print(ex);
    }

    return st;
}
}
```

Γενικές Παρατηρήσεις

Η συνάρτηση *readLine*() είναι εξορισμού συνάρτηση της Java για διάβασμα αλφαριθμητικών από το πληκτρολόγιο. Το διάβασμα δεδομένων από πληκτρολόγιο μπορεί να προκαλέσει σφάλματα-εξαιρέσεις (π.χ. εισάγονται δεδομένα λάθος τύπου, σφάλμα πληκτρολογίου κ.λ.π.) και γιαυτό είναι απαραίτητο να γίνεται πάντα με το μηχανισμό try και catch. Επίσης ο τύπος *BufferedReader* δημιουργεί ένα αντικείμενο Buffer (μνήμης) στο οποίο θα αποθηκευτεί το δεδομένο που διαβάζουμε.

Προκειμένου να μετατρέψουμε τα δεδομένα εισόδου τα οποία είναι πάντα συμβολοσειρά (τύπου String) όπως αναφέραμε, χρησιμοποιούμε τις ακόλουθες συναρτήσεις μετατροπής:

α) Συνάρτηση readint() μετατροπής συμβολοσειράς (String) σε ακέραιο (int)

```
public class metatropi_symboloseiras
{
public static int readint()
{
    return Integer.parseInt(readstring());
}
}
```

β) Συνάρτηση reafloat() μετατροπής συμβολοσειράς (String) σε πραγματικό απλής ακρίβειας (float)

```
public class metatropi_symboloseiras_2
{
public static float readfloat()
{
    return Float.parseFloat(readstring());
```

```
}

}
```

γ) Συνάρτηση readdouble() μετατροπής συμβολοσειράς (String) σε πραγματικό διπλής ακρίβειας(double)

```
public class metatropi_symboloseiras_3

{

public static double readdouble()

{

    return Double.parseDouble(readstring());

}

}
```

Παρατήρηση

Για την υλοποίηση της συνάρτησης readstring είναι απαραίτητη η εντολή *import java.io.** στη αρχή της κλάσης που περιέχει τη συνάρτηση readstring. Η εντολή import είναι αντίστοιχη της include στη C++ και η συγκεκριμένη εισάγει στο πρόγραμμα όλες τις συναρτήσεις του πακέτου *java.io.*

13.2 Εντολή Εξόδου

Η εντολή εξόδου χρησιμοποιείται για να εμφανίσουμε μηνύματα και αποτελέσματα στην οθόνη του Η/Υ. Έχει την ακόλουθη σύνταξη:

```
System.out.println ("σχόλιο"); για να τυπώσουμε ένα σχόλιο

ή

System.out.println (μεταβλητή); για να τυπώσουμε ένα αποτέλεσμα

ή

System.out.println ("σχόλιο"+μεταβλητή); για να τυπώσουμε ένα σχόλιο και ένα αποτέλεσμα
```

14. Εντολή Επιλογής

14.1 Απλή Επιλογή

Στην απλή επιλογή ελέγχεται μια συνθήκη και εφόσον είναι αληθής τότε εκτελείται μια ομάδα εντολών 1, ενώ αν είναι ψευδής τότε εκτελείται μια ομάδα εντολών 2. Έχει την ακόλουθη σύνταξη:

```
if (συνθήκη)
{
   ομάδα εντολών 1;
}
else
{
```

```
  ομάδα εντολών 2;
}
```

14.2 Περιορισμένη Επιλογή

Στην περιορισμένη επιλογή ελέγχεται μια συνθήκη και εφόσον είναι αληθής τότε εκτελείται μια ομάδα εντολών 1 ενώ αν είναι ψευδής τότε δεν εκτελείται τίποτα και το πρόγραμμα συνεχίζει απλώς στην επόμενη εντολή μετά το if. Έχει την ακόλουθη σύνταξη:

```
if (συνθήκη)
{
  ομάδα εντολών 1;
}
```

14.3 Εμφωλευμένη Επιλογή

Στην εμφωλευμένη επιλογή ελέγχεται μια συνθήκη 1 και εφόσον είναι αληθής τότε εκτελείται μια ομάδα εντολών 1. Αν είναι ψευδής τότε ελέγχεται μια συνθήκη 2 και εφόσον είναι αληθής τότε εκτελείται μια ομάδα εντολών 2. Αν είναι ψευδής τότε ελέγχεται μια συνθήκη 3 κ.ο.κ.

Συνολικά ελέγχονται n συνθήκες και εκτελείται μόνο μια ομάδα εντολών. Αν δεν είναι καμία συνθήκη αληθής τότε εκτελείται μια ομάδα εντολών n+1. Έχει την ακόλουθη σύνταξη:

```
if (συνθήκη 1)
```

```
{
   ομάδα εντολών 1;
}
else
   if (συνθήκη 2)
   {
      ομάδα εντολών 2;
   }
   else
      …
      if (συνθήκη n)
      {
       ομάδα εντολών n;
      }
      else
      {
       ομάδα εντολών n+1;
      }
```

Παρατηρήσεις

1. Σε κάθε εντολή επιλογής η συνθήκη είναι μια λογική παράσταση η οποία παίρνει είτε την τιμή αλήθεια (true) είτε την τιμή ψέμα (false). Το 0

αντιστοιχεί στη λογική τιμή αλήθεια και ένας αριθμός 0 αντιστοιχεί στην λογική τιμή ψέμα.

2. Κάθε ομάδα εντολών μπορεί να είναι είτε μια μόνο εντολή ή ένα μπλοκ από εντολές μέσα σε άγκιστρα. Αν είναι μόνο μια εντολή τότε μπορούμε να παραλείψουμε τα άγκιστρα.

14.4 Εντολή Πολλαπλής Επιλογής

Η εντολή πολλαπλής επιλογής χρησιμοποιείται όταν ελέγχουμε την τιμή μιας μεταβλητής (όχι συνθήκης) και ανάλογα με την τιμή αυτή εκτελούμε είτε την ομάδα εντολών 1 είτε την ομάδα εντολών 2 είτε την ομάδα εντολών ν. Αν η μεταβλητή δεν πάρει καμία από τις τιμές 1, 2, .., ν τότε εκτελείται η ομάδα εντολών που βρίσκεται στο default. Η εντολή switch γενικά έχει την ακόλουθη μορφή:

```
switch (μεταβλητή)
{
   case τιμή 1:     ομάδα εντολών 1;
                    break;

   case τιμή 2:     ομάδα εντολών 2;
                    break;
```

```
    case τιμή ν:      ομάδα εντολών ν;
                      break;

    default:          ομάδα εντολών ν+1;
}
```

Σημείωση

Η εντολή break είναι προαιρετική. Όμως ουσιαστικά είναι επιβεβλημένη η χρησιμοποίηση της γιατί αν απουσιάζει από το τέλος εντολών μιας ομάδας τότε εκτελούνται οι εντολές και της επόμενης ομάδας γεγονός βέβαια που δεν έχει νόημα. Συνεπώς η break στην εντολή switch απαγορεύει την εκτέλεση των εντολών των υπολοίπων ομάδων πέρα της ομάδας εντολών που θα εκτελεστεί.

15. Εντολές Επανάληψης

Οι εντολές επανάληψης (ανακύκλωσης) προκαλούν την εκτέλεση μιας εντολής ή μιας ομάδας εντολών μέσα σε άγκιστρα είτε ένα προκαθορισμένο αριθμό επαναλήψεων είτε για όσο χρόνο μια συνθήκη είναι αληθής. Στη Java χρησιμοποιούμε 3 εντολές επανάληψης.

15.1 Εντολή Επανάληψης for

Η εντολή for επαναλαμβάνει μια ομάδα εντολών συνήθως ένα συγκεκριμένο αριθμό φορών αν και μπορεί να χρησιμοποιηθεί ακόμα και για

την επανάληψη μιας ομάδας εντολών για όσο χρόνο μια συνθήκη είναι αληθής. Έχει την ακόλουθη σύνταξη:

```
for (αρχικές τιμές; συνθήκη; μεταβολές)
{
ομάδα εντολών;
}
```

Οι αρχικές τιμές μπορεί να είναι μια ή περισσότερες εντολές που διαχωρίζονται με κόμμα και έχουν στόχο να εκτελεστούν κάποιες εντολές πριν αρχίσει η επανάληψη. Η συνθήκη μπορεί να είναι απλή είτε να συγκροτείται από επιμέρους συνθήκες χωριζόμενες με κόμμα (σύνθετη) και για όσο χρόνο είναι αληθής εκτελείται η ομάδα εντολών ανάμεσα στα άγκιστρα. Μετά από κάθε επανάληψη εκτελούνται οι εντολές που βρίσκονται στο τμήμα μεταβολές οι οποίες έχουν σαν σκοπό να μεταβάλλουν την τιμή της συνθήκης. Στη συνέχεια ελέγχεται πάλι η νέα τιμή της συνθήκης και εφόσον εξακολουθεί να είναι αληθής επαναλαμβάνεται η εκτέλεση της ομάδας εντολών. Όταν η συνθήκη γίνει ψευδής τότε η εντολή for τερματίζεται και το πρόγραμμα συνεχίζει στην επόμενη εντολή μετά την for.

15.2 Εντολή Επανάληψης while

Η εντολή while χρησιμοποιείται όπως και η for για να επαναλάβουμε την εκτέλεση μιας εντολής ή μιας ομάδας εντολών μέσα σε άγκιστρα είτε ένα συγκεκριμένο αριθμό φορών είτε για όσο χρόνο μια συνθήκη είναι αληθής. Πρώτα ελέγχεται η συνθήκη και εφόσον είναι αληθής τότε εκτελείται η ομάδα

εντολών. Για όσο χρόνο η συνθήκη (απλή ή σύνθετη) είναι αληθής (true) τότε επαναλαμβάνεται η εκτέλεση της ομάδας εντολών. Όταν η συνθήκη γίνει ψευδής η εντολή while τερματίζεται και το πρόγραμμα συνεχίζει στην επόμενη εντολή μετά την while. Έχει την ακόλουθη σύνταξη:

```
while (συνθήκη)
{
ομάδα εντολών;
}
```

15.3 Εντολή Επανάληψης do..while

Η εντολή do while είναι αντίστοιχη με την εντολή while με τη διαφορά ότι πρώτα εκτελείται η ομάδα εντολών και μετά ελέγχεται η συνθήκη, δηλαδή η ομάδα εντολών θα εκτελεστεί τουλάχιστον μια φορά ακόμα και αν η συνθήκη είναι ψευδής. Έχει την ακόλουθη σύνταξη:

```
do
{
ομάδα εντολών;
}
while (συνθήκη);
```

16.Λοιπές Εντολές

16.1 Εντολή break

Στις εντολές επανάληψης for, while και do while μπορούμε να παρεμβάλουμε μέσα στην ομάδα εντολών την εντολή break η οποία προκαλεί τη διακοπή της εκτέλεσης των υπολοίπων εντολών της ομάδας και κάνει άμεση έξοδο από την επανάληψη. Έχει την ακόλουθη σύνταξη:

Εντολή επανάληψης

{

break;

}

16.2 Εντολή continue

Η εντολή continue τοποθετείται επίσης μέσα σε μια επανάληψη και προκαλεί τη διακοπή της εκτέλεσης των υπολοίπων εντολών της ομάδας αλλά δεν κάνει έξοδο από το βρόχο, απλώς επιβάλλει επανέλεγχο της συνθήκης. Έχει την ακόλουθη σύνταξη:

εντολή επανάληψης

{

continue;

}

17. Συναρτήσεις (υποπρογράμματα)

Όταν ένα πρόγραμμα μεγαλώνει σε μέγεθος (πλήθος εντολών) τότε το πρόγραμμα αυτό θα πρέπει να διαιρεθεί σε τμήματα. Τα τμήματα αυτά ονομάζονται συναρτήσεις ή μέθοδοι. Τα πλεονεκτήματα της συγγραφής συναρτήσεων αντί για ένα ενιαίο πρόγραμμα είναι τα ακόλουθα:

- διευκολύνεται η ανίχνευση και η διόρθωση λαθών. Γενικά μια συνάρτηση περιλαμβάνει κατά μέσο όρο 10-15 γραμμές κώδικα στις οποίες είναι πιο εύκολος ο εντοπισμός λαθών από ότι σε ένα ενιαίο πρόγραμμα το οποίο περιλαμβάνει συνήθως πάρα πολλές εντολές.
- μια επαναλαμβανόμενη λειτουργία δεν χρειάζεται να πληκτρολογείτε πολλές φορές αλλά μπορεί να εκτελείται καλώντας την ίδια συνάρτηση
- διευκολύνεται η συντήρηση και η επέκταση των προγραμμάτων. Κάθε νέα λειτουργία ισοδυναμεί με μια νέα συνάρτηση, ενώ μια διόρθωση-βελτίωση του προγράμματος σημαίνει τροποποίηση του κώδικα μιας συνάρτησης

Ας δούμε ένα απλό παράδειγμα. Το παρακάτω απλό πρόγραμμα ζητά ένα θετικό ακέραιο αριθμό από τον χρήστη και μετά υπολογίζει το παραγοντικό του. Θα χρησιμοποιήσουμε δύο μεθόδους στο πρόγραμμα, μία που ελέγχει αν ο χρήστης εισήγαγε έναν έγκυρο θετικό ακέραιο και μία άλλη που υπολογίζει το παραγοντικό του. Θα ξεκινήσουμε γράφοντας την κύρια μέθοδο του προγράμματος:

```
public class ypologismos_paragontikou
{
class Factorial
```

```
{
   public static void main(String [] args)
   {
      int n;


      while ((n = getNextInteger()) >= 0)
      System.out.println (factorial(n));
   }
}
}
```

Εκτός των άλλων ο κώδικας αυτός αποδεικνύει ότι οι μέθοδοι καθιστούν δυνατό να σχεδιαστεί η ροή του προγράμματος χωρίς περιττές λεπτομέρειες. Απλά ονομάσαμε δύο μεθόδους getNextInteger() και factorial() χωρίς να ανησυχούμε για την εφαρμογή τους. Μπορούμε να προσθέσουμε τον υπόλοιπο κώδικα σε μικρότερα, ευκολονόητα τμήματα. Ας γράψουμε πρώτα τη μέθοδο factorial:

```
public class factorial
{
long factorial (long n)
{
```

```
  int i;
  long result=1;


  for (i=1; i <= n; i++)
        result *= i;


  return result;
}
}
```

Θα μπορούσαμε να συμπεριλάβουμε αυτό τον κώδικα στη μέθοδο main αλλά είναι πιο εύκολο να κατανοήσουμε τον αλγόριθμο σπάζοντας τον κώδικα σε μικρότερα τμήματα. Είναι επίσης πιο εύκολο να ελέγξουμε και να διορθώσουμε. Μπορούμε να γράψουμε ένα απλό πρόγραμμα που θα μας επιτρέπει να ελέγχουμε τη μέθοδο factorial προτού μας απασχολήσει το αν είναι αποδεκτή και έγκυρη η είσοδος του χρήστη. Ακολουθεί το πρόγραμμα ελέγχου:

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

```
    int n;
    int i;
    long result;



    for (i=1; i <=10; i++)
    {
       result = factorial(i);
       System.out.println (result);
    }
}


static long factorial (int n)
{
    int i;
    long result=1;


    for (i=1; i <= n; i++)
    result *= i;
    return result;
}
```

```
}
```

Παρατηρούμε για άλλη μια φορά ότι τα πάντα στη Java ανήκουν σε κλάσεις. Ας δούμε με μια πιο προσεκτική ματιά τη σύνταξη της μεθόδου:

```
public class long_factorial
}
static long factorial (int n)
{
    int i;
    long result=1;

    for (i=1; i <= n; i++)
        result *= i;
    return result;
}
}
```

Οι μέθοδοι ξεκινούν με μία δήλωση. Αυτό μπορεί να περιλαμβάνει από 3 έως 5 τμήματα. Πρώτα είναι ένας προαιρετικός καθαριστής πρόσβασης (optional access specifier), που μπορεί να είναι γενικός (public), ιδιωτικός

(private) ή προστατευόμενος (protected). Μία γενική μέθοδο μπορούμε να την καλέσουμε από παντού. Μία ιδιωτική μέθοδος μπορεί να χρησιμοποιηθεί μόνο από την κλάση που προσδιορίζεται. Μία προστατευμένη μέθοδος μπορεί να χρησιμοποιηθεί οπουδήποτε εντός του πακέτου που προσδιορίζεται. Οι μέθοδοι που δεν έχουν δηλωθεί σαν γενικές ή ιδιωτικές θεωρούνται προστατευόμενες. Στη συνέχεια αποφασίζουμε αν μια μέθοδος είναι στατική ή όχι. Οι στατικές μέθοδοι έχουν ένα στιγμιότυπο ανά κλάση παρά ένα παράδειγμα ανά αντικείμενο. Όλα τα αντικείμενα μιας κλάσης μοιράζονται ένα αντίγραφο της στατικής μεθόδου. Εξορισμού οι μέθοδοι δεν είναι στατικές. Στη συνέχεια εξειδικεύουμε τον τύπο επιστροφής. Αυτό είναι η τιμή που θα επιστραφεί στην καλούμενη μέθοδο όταν τελειώσουν όλοι οι υπολογισμοί μέσα στη μέθοδο. Για παράδειγμα αν ο τύπος επιστροφής είναι int μπορούμε να χρησιμοποιήσουμε τη μέθοδο όπου χρησιμοποιούμε ένα σταθερό ακέραιο. Αν ο τύπος επιστροφής είναι void τότε δεν επιστρέφεται καμία τιμή. Έπειτα ακολουθεί το όνομα της μεθόδου. Στη συνέχεια έχουμε τις παρενθέσεις. Μέσα στις παρενθέσεις δίνουμε ονόματα και τύπους στα ορίσματα της μεθόδου. Μία μέθοδος μπορεί να έχει κανένα, ένα ή πολλά ορίσματα. Τα ορίσματα αυτά μπορούν να χρησιμοποιηθούν μέσα στη μέθοδο όπως οι τοπικές μεταβλητές. Τέλος το υπόλοιπο της μεθόδου περικλείεται μέσα σε άγκιστρα τα οποία το κάνουν ένα απλό μπλοκ. Το τμήμα μέσα στα άγκιστρα είναι όπως οι κύριες μέθοδοι. Υπάρχουν δηλώσεις μεταβλητών, κώδικας και τέλος μία δήλωση επιστροφής, η οποία στέλνει μια τιμή πίσω στην καλούμενη μέθοδο. Ο τύπος της τιμής αυτής πρέπει να συμφωνεί με τον τύπο που δηλώθηκε στη μέθοδο. Γενικά κάθε γραμμή που μοιάζει με το Text(arg1, arg 2) ή με το text(arg 1) ή text() είναι ένα κάλεσμα μεθόδου.

18. Η Java ως γλώσσα διαδικτύου

Ο ενθουσιασμός που έχει προκληθεί με την χρησιμοποίηση της Java ως γλώσσα ανάπτυξης εφαρμογών στο διαδίκτυο, οφείλεται στην δυνατότητα της γλώσσας αυτής να επιλύει δύο βασικά προβλήματα που έχουν να κάνουν με το περιεχόμενο του διαδικτύου και τα οποία αναλύουμε παρακάτω.

Το περιεχόμενο του WWW είναι παθητικό και στατικό

Οι σελίδες του WWW είναι κατάλληλες για την μετάδοση παθητικών, στατικών πληροφοριών, δηλαδή πληροφοριών που δεν αλλάζουν συχνά (στατικές) και δεν μεταβάλλονται ανάλογα με την αλληλεπίδραση του χρήστη (παθητικό).

Για παράδειγμα, πολλές σελίδες στο Web σου επιτρέπουν την είσοδο στα στοιχεία μίας επιχείρησης όσον αφορά τις τιμές κάποιον μετοχών στο χρηματιστήριο. Τις περισσότερες φορές οι σελίδες αυτές είναι στατικές γιατί δεν ενημερώνονται ταυτόχρονα με τις νέες τιμές των μετοχών όσο ο χρήστης πλοηγείται στην σελίδα. Επίσης, είναι παθητικές γιατί το μόνο που μπορεί να κάνει ο χρήστης είναι να ζητήσει την τιμή μίας μετοχής.

Οι περισσότερες τεχνολογίες ανάπτυξης Web εφαρμογών είναι κατάλληλες για την μετάδοση τέτοιου είδους πληροφοριών. Ωστόσο, η Java επιτρέπει την δημιουργία πιο δυναμικών ιστοσελίδων. Μπορούμε λοιπόν να δημιουργήσουμε ένα διάγραμμα που δείχνει τις τιμές μίας μετοχής το οποίο ενημερώνεται αυτόματα όσο ο χρήστης πλοηγείται στην σελίδα. Η δημιουργία

τέτοιων σελίδων είναι σαφώς ευκολότερη μέσα από την Java παρά από οποιαδήποτε άλλη γλώσσα.

Η ανάγνωση του περιεχομένου του WWW εξαρτάται κάθε φορά από το πρόγραμμα πλοήγησης που χρησιμοποιεί ο χρήστης του Παγκόσμιου Ιστού.

Πριν από την ανάπτυξη της Java, ο σχεδιαστής μίας ιστοσελίδας δεν μπορούσε να ξέρει πιο πρόγραμμα ανάγνωσης ιστοσελίδων (browser) χρησιμοποιεί ο χρήστης. Μπορούσε λοιπόν να χρησιμοποιεί γραφικά και ήχο τα οποία όμως δεν θα μπορούσε να δει ο χρήστης εάν δεν χρησιμοποιούσε το κατάλληλο πρόγραμμα ανάγνωσης ιστοσελίδων.

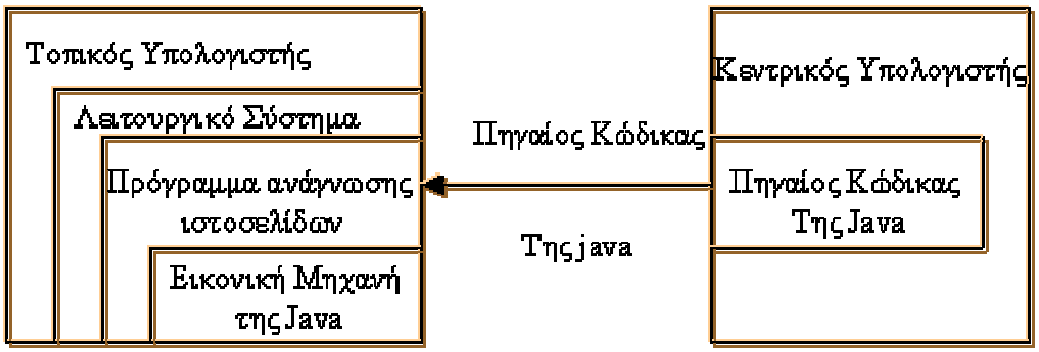
Με την ανάπτυξη όμως της Java το πρόβλημα αυτό επιλύθηκε. Η Java επιτρέπει στον σχεδιαστή της ιστοσελίδας την ανάπτυξη σελίδων που θα παραδοθούν σε όλους τους χρήστες που χρησιμοποιούν ένα συμβατό ως προ την Java πρόγραμμα ανάγνωσης ιστοσελίδων.

19.Πως λειτουργεί η Java

Στο σχήμα 1.1 φαίνεται ο τρόπος με τον οποίο γίνεται η μεταφορά του πηγαίου κώδικα της Java (Java source code) μέσω του διαδικτύου. Υπάρχει ένας κεντρικός υπολογιστής (host) στον οποίο αποθηκεύεται ο κώδικας ενός προγράμματος γραμμένος στην γλώσσα προγραμματισμού Java. Όταν ο χρήστης ενός τοπικού υπολογιστή χρησιμοποιεί το διαδίκτυο για να συνδεθεί με τον κεντρικό υπολογιστή μέσω ενός "συμβατού με την Java" προγράμματος ανάγνωσης ιστοσελίδων (Java- enabledbrowser), ο κώδικας του προγράμματος μεταφέρεται από τον κεντρικό στον τοπικό υπολογιστή.

Ενσωματωμένη μέσα σε ένα συμβατό ως προς την Java πρόγραμμα ανάγνωσης ιστοσελίδων βρίσκεται η εικονική μηχανή της Java (Virual Machine), γνωστή και με την ονομασία Διερμηνευτής Χρόνου Εκτέλεσης που αναλύθηκε σε προηγούμενη ενότητα.

Σχήμα 1: Μεταφορά του πηγαίου κώδικα ενός προγράμματος της Java μέσω του διαδικτύου.



20. Τα εργαλεία της Java

Για να γράψει κανείς μία εφαρμογή ή μίνι - εφαρμογή στην Java, χρειάζεται εκτός από την ίδια την γλώσσα και τα εργαλεία με τα οποία θα γράψει, θα δοκιμάσει και θα μεταγλωττίσει το πρόγραμμα. Κατά καιρούς έχουν αναπτυχθεί πολλά περιβάλλοντα για την εκτέλεση των προγραμμάτων σε Java. Ωστόσο αυτό που κυρίως χρησιμοποιείται είναι το JDK που αναπτύχθηκε από

την Sun Microsystems και το οποίο είναι διαθέσιμο στην ηλεκτρονική διεύθυνση http://www.java.sun.com/.

20.1 Ο διερμηνευτής χρόνου εκτέλεσης

Ο διερμηνευτής χρόνου εκτέλεσης (Interpreter), γνωστός και ως Java, είναι ένα εργαλείο το οποίο χρησιμοποιείται για την εκτέλεση προγραμμάτων και εφαρμογών της Java. Οι μίνι- εφαρμογές της Java απαιτούν την υποστήριξη ενός Web προγράμματος περιήγησης για την εμφάνιση τους. Μπορεί να είναι ένα αυτόνομο τμήμα λογισμικού ή μέρος μίας μεγάλης εφαρμογής, όπως είναι το Netscape. Επιτρέπει στο χρήστη να κάνει πολλά πράγματα, όπως είναι η φόρτωση αρχείων κλάσεων και η ερμηνεία δυαδικού κώδικα (δηλαδή αρχείων σε μορφή bytecode). Πολλές φορές λειτουργεί και ως μεταγλωττιστής ο οποίος μεταγλωττίζει τον δυαδικό κώδικα Java σε κώδικα μηχανής.

Έχει την δυνατότητα να εκτελεί τόσο προγράμματα με υποστήριξη μόνο κειμένου, όσο και εφαρμογές Java με υποστήριξη γραφικών, αλλά όχι μίνι-εφαρμογές.

Η σύνταξη για την χρήση του διερμηνευτή χρόνου εκτέλεσης είναι:

java Επιλογές Όνομα_κλάσης Ορίσματα

Το όρισμα Όνομα_κλάσης προσδιορίζει το όνομα της κλάσης που θέλουμε να εκτελεστεί. Εάν η κλάση βρίσκεται σε ένα πακέτο, πρέπει να προσδιοριστεί πλήρως το όνομα της. Για παράδειγμα, εάν θέλετε να εκτελέσετε μία κλάση με το όνομα Solvent η οποία βρίσκεται σε ένα πακέτο με όνομα Equations, θα πρέπει να καλέσετε τον διερμηνευτή ως εξής:

java Equations.Solvelt

Όταν ο διερμηνευτής της Java εκτελεί μία κλάση, αυτό που κάνει στην πραγματικότητα είναι να εκτελεί την μέθοδο main() της κλάσης. Ο διερμηνευτής τερματίζει όταν η μέθοδος main() και οποιαδήποτε νήματα δημιουργήθηκαν από αυτήν ολοκλήρωσαν την εκτέλεση τους. Η μέθοδος main() δέχεται μία λίστα ορισμάτων τα οποία μπορούν να χρησιμοποιηθούν για τον έλεγχο του προγράμματος.

 Το γεγονός ότι ο διερμηνευτής χρόνου εκτέλεσης της Java εκτελεί στην πραγματικότητα την μέθοδο main() όταν εκτελεί μία κλάση, αποκαλύπτει έναν από τους λόγους για τους οποίους δεν μπορείτε να τρέχετε μίνι - εφαρμογές χρησιμοποιώντας τον διερμηνευτή χρόνου εκτέλεσης.

20.2 Μεταγλωττιστής

Ο μεταγλωττιστής της Java (javac) χρησιμοποιείται για την μεταγλώττιση των αρχείων πηγαίου κώδικα σε εκτελέσιμες κλάσεις (σε μορφή bytecode). Στην Java, τα αρχεία πηγαίου κώδικα έχουν επέκταση .java και είναι αρχεία ASCII, παρόμοια με τα αρχεία πηγαίου κώδικα άλλων γλωσσών προγραμματισμού όπως είναι η C++. Τα εκτελέσιμα αρχεία έχουν επέκταση .class και αντιπροσωπεύουν μία κλάση της Java στην χρησιμοποιήσιμη μορφή της.

Τα bytecode της Java είναι ένα ειδικό σύνολο εντολών μηχανής, οι οποίες δεν απευθύνονται αποκλειστικά σε έναν συγκεκριμένο επεξεργαστή ή σύστημα υπολογιστή. Ένας συγκεκριμένος, ως προς το σύστημα διερμηνευτής, εκτελεί τις οδηγίες των bytecode. Ο διερμηνευτής αυτός λέγεται και εικονική μηχανή της Java.

Ο μεταγλωττιστής επιτρέπει την ύπαρξη μίας μόνο public κλάσης. Είναι απαραίτητη η κοινή ονομασία του αρχείου και της κλάσης.

Η σύνταξη της εντολής είναι:

javac Επιλογές Όνομα Αρχείου

Το όρισμα Όνομα Αρχείου καθορίζει το όνομα του αρχείου με τον πηγαίο κώδικα που θέλετε να μεταγλωττίσετε. Ο μεταγλωττιστής θα παράγει αρχεία κλάσεων σε μορφή bytecode για όλες τις κλάσεις που ορίζονται μέσα στο αρχείο αυτό. Εάν μεταγλωττίζετε την κλάση Α η οποία παράγεται από την κλάση Β και η κλάση Β δεν έχει μεταγλωττιστή ακόμα, τότε ο μεταγλωττιστής παρατηρεί την εξάρτηση αυτή και μεταγλωττίζει και τις δύο κλάσεις. Έτσι, με την εκτέλεση της εντολής javac BigBird δημιουργείται μία κλάση με το όνομα BigBird.class που αποθηκεύεται σε ένα κοινό κατάλογο με το πηγαίο αρχείο.

Το όρισμα Επιλογές του μεταγλωττιστή καθορίζει επιλογές που σχετίζονται με το πως δημιουργούνται τα εκτελέσιμα αρχεία κλάσεων της Java από τον μεταγλωττιστή.

Η επιλογή -d καθορίζει τον αρχικό κατάλογο στον οποίο αποθηκεύονται οι μεταγλωττισμένες κλάσεις. Αυτό είναι σημαντικό επειδή πολλές φορές οι κλάσεις οργανώνονται σε ιεραρχική δομή καταλόγων. Με την επιλογή -d, η δομή καταλόγων θα δημιουργηθεί κάτω από τον κατάλογο που προσδιορίζει η επιλογή.

Η επιλογή -g αναγκάζει τον μεταγλωττιστή να παράγει πίνακες αποσφαλμάτωσης για τις κλάσεις της Java. Οι πίνακες αποσφαλμάτωσης χρησιμοποιούνται από τον αποσφαλματωτή της Java και περιέχουν πληροφορίες όπως τοπικές μεταβλητές και αριθμοί γραμμών.

 Ο μεταγλωττιστής μπορεί να εκτελέσει την δουλειά του ακόμα και αν μόνο οι μεταγλωττισμένες εκδοχές των κλάσεων είναι διαθέσιμες. Δεν χρειάζεται να υπάρχει κώδικας για όλα τα αντικείμενα. Οι κλάσεις της Java έχουν όλες τις πληροφορίες που είναι διαθέσιμες στα αρχεία κώδικα.

20.3 Αρχεία JAR (Java Archive Files)

Μπορείτε να χρησιμοποιήσετε τα αρχεία JAR για να συμπιέσετε και να εσωκλείσετε ένα μεγάλο αριθμό αρχείων έτσι ώστε να μπορείτε να τα φορτώσετε όλα μαζί με μίας σε ένα μίνι πακέτο, πράγμα που επιταχύνει σημαντικά την διαδικασία της φόρτωσης. Για να δημιουργήσετε ένα αρχείο JAR, χρησιμοποιείστε το αρχείο jar.exe. Για παράδειγμα έστω ότι η μίνι-εφαρμογή canvaser.java δημιουργεί δύο αρχεία κλάσης κατά την μεταγλώττιση:canvaser.class και BoxCanvas.class. Μπορείτε να συμπιέσετε (Zip) και τα δύο αρχεία αυτά και να τοποθετήσετε σε ένα αρχείο JAR έτσι ώστε η φόρτωσή τους να γίνεται μαζί. Για να το κάνετε αυτό χρησιμοποιήστε το Jar.exe για να δημιουργήσετε το canvaser.jar ως εξής:

*C:\java-1\canvaser>jar cvf canvaser.jar**

Το σύμβολο " * " στο τέλος της παραπάνω πρότασης σημαίνει ότι θα συμπιέσετε όλα τα αρχεία του τρέχοντος καταλόγου μέσα σε αυτό το αρχείο JAR, ενώ η συμβολοσειρά cvf δείχνει τις επιλογές που θέλετε να χρησιμοποιήσετε με το αρχείο jar.exe. Οι επιλογές αυτές παρατίθενται στον ακόλουθο πίνακα.

Πίνακας 1.1: Επιλογές που έχετε στη διάθεσή σας με το jar.exe

Επιλογή	**Λειτουργία**
C	Δημιουργεί μία νέα ή άδεια αρχειοθέτηση πάνω στη βασική έξοδο
T	Παραθέτει τον πίνακα περιεχομένων από τη βασική έξοδο
x [αρχείο]	Εξάγει όλα τα αρχεία ή απλώς τα αρχεία που έχουν όνομα, από τη βασική είσοδο. Αν το αρχείο αφαιρείται, τότε εξάγονται όλα τα αρχεία, διαφορετικά εξάγεται μόνο το καθορισμένο αρχείο (ή τα καθορισμένα αρχεία).
F [jar - αρχείο]	Το δεύτερο όρισμα καθορίζει ένα αρχείο σε επεξεργασία. Στη περίπτωση της δημιουργίας, το εν λόγω όρισμα αναφέρεται στο όνομα του αρχείου JAR που πρόκειται να δημιουργηθεί (στη θέση της βασικής εξόδου). Για πίνακα, το δεύτερο όρισμα προσδιορίζει το αρχείο JAR που πρόκειται να παρατεθεί σε λίστα ή να εξαχθεί.
V	Δημιουργεί μία έξοδο (αρκετά φορτωμένη) στο stderr.

20.4 Το μονοπάτι της κλάσης

Για να μπορεί η Java να χρησιμοποιήσει μία κλάση, πρέπει να μπορεί να την βρει στο σύστημα αρχείων. Αλλιώς, θα εκδοθεί μήνυμα ότι η κλάση δεν υπάρχει. Η Java χρησιμοποιεί δύο πράγματα για να βρίσκει τις κλάσεις: το

όνομα του ίδιου του πακέτου και τους καταλόγους που αναφέρονται στην μεταβλητή CLASSPATH.

Τα ονόματα πακέτων αντιστοιχούν σε ονόματα καταλόγων του συστήματος αρχείων, οπότε η κλάση java.applet.Appplet θα βρίσκεται πραγματικά στον κατάλογο java.

Η java ψάχνει για αυτούς τους καταλόγους με την σειρά, μέσα στους καταλόγους που αναφέρονται στην μεταβλητή CLASSPATH. Όταν η java ψάχνει για μία κλάση την οποία έχετε αναφέρει στον πηγαίο κώδικα σας, ψάχνει για το όνομα του πακέτου και της κλάσης σε κάθε ένα από τους καταλόγους αυτούς και επιστρέφει λάθος εάν δεν μπορεί να βρει το αρχείο της κλάσης.

20.5 Βοήθημα εμφάνισης μίνι-εφαρμογών (Applet Viewer)

Οι δημιουργοί εφαρμογών της Java έχουν έναν ακόμη τρόπο για να τρέχουν τις μίνι - εφαρμογές, ο οποίος δεν απαιτεί την χρήση ενός Web προγράμματος περιήγησης. Ο τρόπος αυτός είναι το βοήθημα εμφάνισης μίνι - εφαρμογών (applet viewer) της java, το οποίο εξυπηρετεί σαν ένας στοιχειώδης μηχανισμός ελέγχου των μίνι - εφαρμογών που γράφονται με την java.

Αν και το βοήθημα εμφάνισης μίνι - εφαρμογών λογικά καταλαμβάνει την θέση ενός Web προγράμματος περιήγησης, λειτουργεί πολύ διαφορετικά από τα web προγράμματα περιήγησης. Κατ΄ αρχήν λειτουργεί πάνω σε HTML έγγραφα, αλλά ψάχνει μόνο για τις ενσωματωμένες ετικέτες <Applet> και αγνοεί όλο τον υπόλοιπο κώδικα που υπάρχει στο έγγραφο. Κάθε φορά που το βοήθημα εμφάνισης μίνι - εφαρμογών συναντά μία ετικέτα <Applet> μέσα σε ένα HTML έγγραφο, ανοίγει ένα ξεχωριστό παράθυρο, το οποίο περιέχει την αντίστοιχη μίνι - εφαρμογή.

Το μόνο μειονέκτημα είναι ότι δεν δείχνει πως θα τρέχει η μίνι - εφαρμογή όταν θα περιοριστεί μέσα στα όρια μίας πραγματικής ιστοσελίδας.

Η σύνταξή του είναι:

appletviewer Επιλογές URL

Το όρισμα URL καθορίζει το URL του εγγράφου που περιέχει μία HTML σελίδα σε μία ενσωματωμένη μίνι - εφαρμογή της Java.

20.6 Αποσφαλματωτής

Ο απασφαλματωτής της java (jdb) επιτρέπει την αποσφαλμάτωση των προγραμμάτων που γράφονται με την java. Παρέχει τις εξής λειτουργίες αποσφαλμάτωσης: τον ορισμό των σημείων διακοπής, δηλαδή τον καθορισμό της γραμμής του κώδικα στην οποία σταματά η εκτέλεση του προγράμματος και την βηματική εκτέλεση του κώδικα, δηλαδή την εκτέλεση του κώδικα γραμμή προς γραμμή.

Πριν χρησιμοποιηθεί ο αποσφαλματωτής θα πρέπει πρώτα να μεταγλωττιστεί ο κώδικας.

Η σύνταξη του είναι:

jdb Επιλογές <Όνομα Κλάσης>

Το όρισμα Όνομα Κλάσης είναι προαιρετικό και καθορίζει το όνομα της κλάσης που θα εκτελεστεί.

20.7 Γεννήτρια τεκμηρίωσης (JavaDoc)

Η γεννήτρια τεκμηρίωσης της Java (javadoc) είναι ένα χρήσιμο εργαλείο για την παραγωγή τεκμηρίωσης απευθείας από τον πηγαίο κώδικα. Αναλύει τα πηγαία αρχεία και παράγει HTML σελίδες βασιζόμενη στις δηλώσεις και στα σχόλια.

Η σύνταξη της είναι:

javadoc Επιλογές Όνομα Αρχείου

Το όρισμα Όνομα Αρχείου καθορίζει είτε ένα πακέτο, είτε ένα αρχείο πηγαίου κώδικα της Java.

Η γεννήτρια τεκμηρίωσης υποστηρίζει ειδικές ετικέτες για την πρόσθεση επιπλέον πληροφοριών στα παραγόμενα HTML έγγραφα. Όλες οι ετικέτες ξεκινούν με το σύμβολο @ το οποίο και πρέπει να εμφανίζεται στην αρχή κάθε γραμμής. Ένα παράδειγμα πηγαίου κώδικα που χρησιμοποιεί ετικέτες τεκμηρίωσης κλάσεων είναι και το ακόλουθο:

```
/**
 * @see Object
 * @see gemology.Rock
 *@version 2.0 Dec 5, 1996
 * @author Brett Weir
 */
```

20.8 Diassembler αρχείων κλάσεων

Ο diassembler αρχείων κλάσεων της java (javap) χρησιμοποιείται για την "αποσυναρμολόγηση" ενός αρχείου κλάσης, πράγμα που σημαίνει ότι το εκτελέσιμο αρχείο κλάσης αναλύεται σε μία λίστα δημόσιων δεδομένων, μεθόδων ή οδηγιών μορφής bytecode. Ο diassembler αρχείων κλάσεων είναι χρήσιμος όταν δεν έχετε τον πηγαίο κώδικα για μία κλάση, αλλά θέλετε να μάθετε κάτι για τον τρόπο υλοποίησής του.

21. Δομές Δεδομένων στη Java

Όπως είναι γνωστό οι δομές δεδομένων χρησιμοποιούνται για την συντήρηση των πληροφοριών που χρειάζονται σε ένα πρόγραμμα. Στον προγραμματισμό με την Java η χρησιμοποίηση δομών δεδομένων είναι απαραίτητη και μπορείτε να χρησιμοποιήσετε είτε υπάρχουσες μορφές δομών δεδομένων είτε να δημιουργήσετε δικές σας.

22. Αναδρομή (Recursion)

Πολλοί πιστεύουν ότι δεν είναι δυνατός ο ορισμός ενός αντικειμένου με βάση το ίδιο το αντικείμενο. Ωστόσο όμως, η έστω και μερική ερμηνεία ενός αντικειμένου με βάση το ίδιο το αντικείμενο είναι μια σημαντική τεχνική. Η επαναληπτική ερμηνεία χρησιμοποιεί την ιδέα ή το αντικείμενο που ορίζεται, ως μέρος της ερμηνείας. Για παράδειγμα, μια πρόταση μπορεί να είναι δύο προτάσεις συνδεδεμένες με μια συνδετική λέξη. Μια πρόταση "while" στην Java μπορεί να δημιουργηθεί από τη λέξη αυτή και άλλες προτάσεις.

Οι αναδρομικές ερμηνείες μπορούν να περιγράψουν περίπλοκες καταστάσεις μέσα σε λίγες μόνο λέξεις. Η ερμηνεία των παραπάνω λέξεων θα ήταν πολύ δύσκολη χωρίς τη χρήση των αναδρομικών ερμηνειών.

Η αναδρομή μπορεί να χρησιμοποιηθεί και ως προγραμματιστική τεχνική. Η αναδρομική υπό-ρουτίνα είναι αυτή που καλεί τον εαυτό της, είτε άμεσα είτε έμμεσα. Όταν λέμε ότι μια υπό-ρουτίνα καλεί τον εαυτό της άμεσα εννοούμε ότι η ερμηνεία της περιέχει μια πρόταση κλήσης υπό-ρουτίνας που καλεί την υπό-ρουτίνα που ορίζεται. Όταν λέμε ότι μια υπό-ρουτίνα καλεί τον εαυτό της έμμεσα εννοούμε ότι καλεί μια άλλη υπό-ρουτίνα που με τη σειρά της καλεί την πρώτη υπό-ρουτίνα. Η επαναληπτική υπό-ρουτίνα μπορεί να ορίσει μια περίπλοκη εργασία μέσα σε λίγες γραμμές κώδικα.

Ας μελετήσουμε το παράδειγμα του αλγορίθμου δυαδικής αναζήτησης, Η δυαδική αναζήτηση χρησιμοποιείται για την εύρεση μιας συγκεκριμένης τιμής σε μια ταξινομημένη λίστα στοιχείων. Η ιδέα είναι ο έλεγχος των στοιχείων της λίστας που βρίσκονται στη μέση και η σύγκρισή τους με μια συγκεκριμένη τιμή. Αν οι τιμές ταιριάζουν τότε το πρόβλημα λύθηκε. Αν η τιμή που αναζητούμε είναι μεγαλύτερη τότε γίνεται αναζήτηση στο μεγαλύτερο μισό, αλλιώς στο μικρότερο. Αυτή είναι μια επαναληπτική περιγραφή και μπορεί να γραφεί μια επαναληπτική υπό-ρουτίνα για να εφαρμοστεί.

Πριν από αυτό όμως, πρέπει να λάβετε υπόψη σας δύο στοιχεία: Ο αλγόριθμος αυτός ξεκινά με τον έλεγχο του κεντρικού στοιχείου. Τι θα συμβεί όμως στην περίπτωση που το στοιχείο αυτό είναι κενό ; Τότε δεν μπορεί να εκτελεστεί ο αλγόριθμος. Επομένως, η ύπαρξη μη κενών λιστών είναι απαραίτητη προϋπόθεση για τις αναδρομικές μεθόδους.

Το δεύτερο στοιχείο έχει να κάνει με τον αριθμό των ορισμάτων που θα υπάρξουν στην υπό-ρουτίνα. Στην επαναληπτική υπό-ρουτίνα πρέπει να είναι καθορισμένος ο αριθμός των στοιχείων που θα γίνει αναζήτηση και όχι όλα τα

στοιχεία του πίνακα. Αυτό δείχνει ότι ένα βασικό στοιχείο για την επαναληπτική επίλυση ενός προβλήματος είναι η γενίκευσή του.

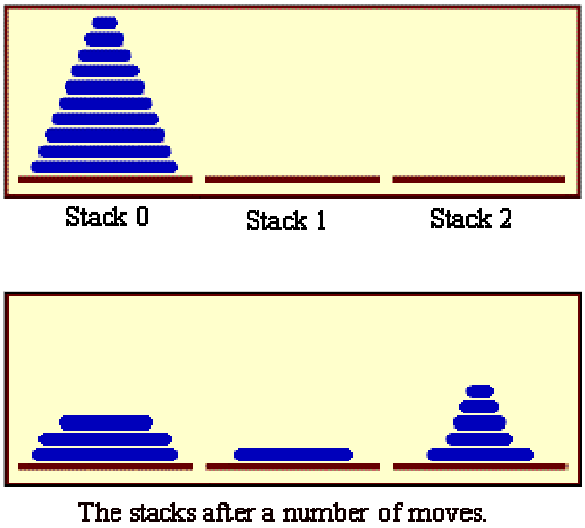
Ο επαναληπτικός αλγόριθμος δυαδικής αναζήτησης για μια συγκεκριμένη τιμή σε ένα πίνακα ακεραίων είναι :

```
Public class binarysearch
{
static int binarySearch(int[] A, int loIndex, int hiIndex, int value) {
 if (loIndex > hiIndex) {
   return -1;
 }
else {
 int middle = (loIndex + hiIndex) / 2;
if (value == A[middle]) return middle;
else if (value < A[middle])
return binarySearch(A, loIndex, middle - 1, value);
else
 return binarySearch(A, middle + 1, hiIndex, value);
 } } // end binarySearch().
 }
```

Σε αυτή την ρουτίνα, η παράμετρος loIndex και hiIndex προσδιορίζουν το μέρος του πίνακα που θα εξεταστεί. Για την εξέταση όλου του πίνακα απλά καλείτε την binarySearch (A,0,A.lenght -1, value). Στις δύο βασικές περιπτώσεις - όπου δεν υπάρχουν στοιχεία στον πίνακα και η τιμή που αναζητάτε είναι στο κέντρο του πίνακα - η υπό-ρουτίνα μπορεί να επιστρέψει μια απάντηση αμέσως χωρίς τη χρήση της αναδρομής. Στις άλλες περιπτώσεις χρησιμοποιεί επαναληπτική κλήση για τον προσδιορισμό των απαντήσεων.

Για τη σωστή λοιπόν λειτουργία των επαναληπτικών κλήσεων πρέπει να υπάρχουν δύο βασικές κλάσεις που να μπορούν να χειριστούν χωρίς επαναληπτική κλήση. Και όταν εφαρμόζεται η επανάληψη κατά την επίλυση ενός προβλήματος, θα πρέπει να εφαρμόζεται σε ένα πρόβλημα μικρότερο του αρχικού. Η ιδέα είναι ότι αφού είναι δυνατή η επίλυση μικρών προβλημάτων άρα μπορείτε να περιορίσετε το αρχικό μέγεθος των προβλημάτων. Ένα σημαντικό λάθος κατά τη δημιουργία επαναληπτικών υπό-ρουτινών είναι η καταπάτηση ενός από τους παραπάνω δύο κανόνες.

Ένα άλλο πρόβλημα που μπορεί εύκολα να λυθεί με τη μέθοδο των επαναληπτικών υπό-ρουτινών είναι το "The Towers of Hanoi". Το πρόβλημα καλεί μια στοίβα από δίσκους διαφορετικών μεγεθών, τοποθετημένους τον ένα πάνω στον άλλο. Μόνο ένας δίσκος μπορεί να μετακινηθεί κάθε φορά και δεν μπορεί αν τοποθετηθεί δίσκος πάνω από μικρότερο. Στην παρακάτω εικόνα φαίνεται η τοποθέτηση των δίσκων μετά από δέκα μετακινήσεις:



The stacks after a number of moves.

Το πρόβλημα είναι η μετακίνηση των δέκα δίσκων από τη μια στοίβα 0 στη στοίβα 1. Η στοίβα 2 μπορεί να χρησιμοποιηθεί ως βοηθητική στοίβα. Αυτό μπορεί να επιτευχθεί με τον περιορισμό του προβλήματος. Αυτό ακριβώς είναι που κάνει η αναδρομική υπό-ρουτίνα : Προσδιορίζεται η στοίβα πηγή και

προορισμός. Η λύση είναι η μετακίνηση ενός μόνου δίσκου σε κάθε βήμα. Μια εκδοχή της υπό-ρουτίνας που περιγράφει την αναδρομική μέθοδο είναι :

```
Public class TowersOfHanoi
{
void TowersOfHanoi(int disks, int from, int to, int spare) {

 if (disks == 1) {

System.out.println("Move a disk from stack number " + from + " to stack number " + to);
}
else {
TowersOfHanoi(disks-1, from, spare, to);
System.out.println("Move a disk from stack number " + from + " to stack number " + to);
TowersOfHanoi(disks-1, spare, to, from); } }
 }
```

Μια πιο απλή εφαρμογή είναι η χρησιμοποίηση ενός αναδρομικού αλγορίθμου για την ταξινόμηση ενός πίνακα. Ο αναδρομικός αλγόριθμος Quicksort είναι ένας από τους πιο γρήγορους αλγορίθμους ταξινόμησης.

Ο αλγόριθμος αυτός βασίζεται σε μια απλή, αλλά έξυπνη ιδέα: Δοσμένης μιας λίστας στοιχείων, επιλέξτε ένα στοιχείο από τη λίστα. Αυτό το στοιχείο καλείται pivot . Μετακινείστε όλα τα στοιχεία που είναι μικρότερα στην αρχή

της λίστας και τα μεγαλύτερα στο τέλος. Έτσι έχετε τοποθετήσει το pivot στη μέση των δύο ομάδων. Η διαδικασία αυτή καλείται QuicksortStep.

Η διαδικασία αυτή δεν είναι επαναληπτική. Χρησιμοποιείται ως υπό-ρουτίνα της Quicksort. Η ταχύτητα της εξαρτάται από την ταχύτητα εκτέλεσης της quicksortStep.

```
public class QuicksortStep
{
static int QuicksortStep(int[] A, int lo, int hi) {
 int pivot = A[lo];
while (hi > lo) {
while (hi > lo && A[hi] > pivot)
{
 hi--;
 }
 if (hi == lo)
break;
 A[lo] = A[hi];
 lo++;
 while (hi > lo && A[lo] < pivot) {
  lo++;
 }
 if (hi == lo) break;
 A[hi] = A[lo];
```

```
hi--;
}
A[lo] = pivot; return lo;
} // τέλος της QuicksortStep
}
```

Με την υπό-ρουτίνα αυτή, η Quicksort είναι εύκολη. Ο αλγόριθμος Quicksort αποτελείται από την εφαρμογή της QuicksortStep στην λίστα και έπειτα εφαρμογή της Quicksort στα στοιχεία που βρίσκονται αριστερά του pivot και στα δεξιά του. Αν η λίστα αυτή έχει μόνο ένα στοιχείο τότε δεν χρειάζεται η εφαρμογή του αλγορίθμου Quicksort .

```
public class Quicksort
{
static void Quicksort(int[] A, int lo, int hi) {
if (hi <= lo) {
return;
}
else {
int pivotPosition = quicksortStep(A, lo, hi);
quicksort(A, lo, pivotPosition - 1);
quicksort(A, pivotPosition + 1, hi);
} }
}
```

Παρακάτω παρουσιάζεται μια μίνι εφαρμογή που δείχνει ένα πλέγμα μικρών τετραγώνων. Τα γκρι τετράγωνα είναι "γεμάτα", ενώ τα άσπρα "κενά".

Γι' αυτή την μίνι εφαρμογή ορίζουμε το "blob" να αποτελείται από "γεμάτο" τετράγωνο και από "γεμάτα" τετράγωνα που μπορούν να πλησιάσουν από αυτό με τη μετακίνηση άλλων "γεμάτων" τετράγωνων. Αν κάνετε κλικ σε κάποιο από τα "γεμάτα" τετράγωνα, ο υπολογιστής θα μετρήσει τα τετράγωνα που περιέχει το "blob" και θα αλλάξει το χρώμα τους σε κόκκινο. Για να δημιουργήσετε ένα καινούριο πρότυπο απλά πατήστε το κουμπί "New Blobs" . Το αναδυόμενο μενού δίνει το ποσοστό των τετραγώνων που θα γεμίσουν στο νέο πρότυπο. Όσο περισσότερα γεμάτα τετράγωνα τόσο μεγαλύτερο το "blob". Για να δείτε το πλήθος των διαφορετικών προτύπων blob απλά πατήστε το κουμπί "Count the Blobs".

Ο κώδικας του παραδείγματος αυτού είναι: Blobs

Η επανάληψη χρησιμοποιείται σε αυτή τη μίνι εφαρμογή για τον προσδιορισμό του αριθμού των τετραγώνων στο "blob". Χωρίς την επανάληψη αυτό θα ήταν δύσκολο. Η επανάληψη το κάνει σχετικά εύκολο, αλλά χρειάζεται μια νέα τεχνική. Τα δεδομένα για το πλέγμα των τετραγώνων αποθηκεύονται σε ένα δισδιάστατο πίνακα τιμών boolean:

boolean [] [] filled;

Η τιμή του κάθε στοιχείου του πίνακα filled είναι αληθής αν το αντίστοιχο τετράγωνο είναι "γεμάτο". Η μίνι εφαρμογή χρησιμοποιεί την επαναληπτική μέθοδο getBlobSize() για την αρίθμηση των "γεμάτων" τετραγώνων που περιέχει το "blob". Αν στην συγκεκριμένη θέση δεν υπάρχει "γεμάτο" τετράγωνο, τότε η απάντηση είναι 0. Η εφαρμογή του κώδικα είναι :

```
public class getBlobSize
{
int getBlobSize(int r, int c) {
```

```
if (r < 0 || r >= rows || c < 0 || c >= columns) {
 return 0; }
 if (filled[r][c] == false) {
 return 0; }
 int size = 1;
size += getBlobSize(r-1,c);
 size += getBlobSize(r+1,c);
 size += getBlobSize(r,c-1);
size += getBlobSize(r,c+1);
 return size; }
 }
```

Δυστυχώς αυτή η ρουτίνα θα μετρήσει δύο φορές το ίδιο τετράγωνο. Για την επίλυση αυτού του προβλήματος απλά σημειώνεται κάθε τετράγωνο που έχει μετρηθεί. Η τεχνική των "μαρκαρισμένων" στοιχείων χρησιμοποιείται κατά κόρον στους επαναληπτικούς αλγορίθμους. Η σωστή έκδοση της getBlobSize() είναι:

```
Public class getBlobSize2
{
int getBlobSize(int r, int c) {
 if (r < 0 || r >= rows || c < 0 || c >= columns) {
 return 0; }
 if (filled[r][c] == false || visited[r][c] == true) {
 return 0; }
 visited[r][c] = true;
```

```
 int size = 1;
size += getBlobSize(r-1,c);
size += getBlobSize(r+1,c);
size += getBlobSize(r,c-1);
size += getBlobSize(r,c+1);
return size; } //τέλος της getBlobSize()
}
```

Σε αυτή τη μίνι εφαρμογή, η μέθοδος αυτή η χρησιμοποιείται για τον καθορισμό του μεγέθους του "blob" όταν ο χρήστης κάνει κλικ στο τετράγωνο. Μετά την εκτέλεση της μεθόδου getBlobSize() όλα τα τετράγωνα είναι ακόμη ορατά. Η μέθοδος paint() ζωγραφίζει κόκκινα τα τετράγωνα που κάνουν ορατό το "blob". Ο τρόπος που δουλεύει η μέθοδος αυτή είναι απλός:

```
public class countBlobs
{
void countBlobs() {
 int count = 0;
for (int r = 0; r < rows; r++)
  for (int c = 0; c < columns; c++)
visited[r][c] = false;
 for (int r = 0; r < rows; r++)
 for (int c = 0; c < columns; c++)
  { if (getBlobSize(r,c) > 0)
     count++;
```

```
  }
   repaint();
setText("The number of blobs is " + count); }
}
```

23. Απαρίθμηση (Enumeration)

Πρόκειται ουσιαστικά για ένα σύστημα διασύνδεσης που παρέχει μεθόδους για την απαρίθμηση αλληλουχικών δεδομένων. Ορίζει τον τρόπο ανάκτησης μιας σειράς στοιχείων και χρησιμοποιείται πάντοτε από ήδη υπάρχουσες δομές δεδομένων. Η διασύνδεση αυτή ουσιαστικά παρέχει δύο μεθόδους για τη διαχείριση των δεδομένων. Η μία είναι η

```
public abstract boolean hasMoreElements( );
```

που χρησιμοποιείται για να προσδιοριστεί αν μια δομή δεδομένων έχει περισσότερα στοιχεία. Και η

```
public abstract Object  nextElement( );
```

που χρησιμοποιείται για την ανάκτηση του επόμενου στοιχείου μιας απαριθμημένης λίστας. Και οι δύο αυτές μέθοδοι χρησιμοποιούνται σε συνδυασμό. Για παράδειγμα:

```
while (e.hasMoreElements ( )  )  {
  Object o = e.nextElement( );
  System.out.println( o);
 }
```

24. Βασικά Στοιχεία των Δομών Δεδομένων

Οι δομές δεδομένων συγκαταλέγονται ανάμεσα σε εκείνα τα συστατικά της επιστήμης της πληροφορικής που η χρησιμότητά τους είναι μεγάλη. Ακόμη και οι απλές μίνι εφαρμογές στην Java, μπορούν να χρησιμοποιούν τις διάφορες μορφές δομών δεδομένων για την καλύτερη αποθήκευση και διαχείριση των πληροφοριών. Για παράδειγμα, μια δομή δεδομένων σε μορφή ενός πίνακα μπορεί να χρησιμοποιηθεί σε μια απλή μίνι εφαρμογή που δημιουργεί το εφέ της κίνησης με την εναλλαγή διάφορων εικόνων.

Παρόλο που οι πίνακες αποτελούν την πιο απλή μορφή δομής δεδομένων, ωστόσο όμως η απόδοσή τους είναι άριστη και στην αποθήκευση πιο περίπλοκων δεδομένων.

Η Java με τη βιβλιοθήκη java.util παρέχει ένα σύνολο δομών δεδομένων, που μπορούν να ανταποκριθούν στις πιο υψηλές απαιτήσεις για καλύτερη οργάνωση και διαχείριση των διάφορων πληροφοριών. Επιπλέον, σας δίνει τη δυνατότητα να γράψετε τις δικές σας δομές δεδομένων σύμφωνα με τις ανάγκες σας.

25. Δομές Δεδομένων που προέρχονται από την Java

Η Java παρέχει ορισμένες πολύ καλές δομές δεδομένων που δημιουργούνται από το σύστημα διασύνδεσης Enumeration και πέντε κλάσεις :

- BitSet
- Vector
- Stack/Queues
- Dictionary
- Hashtable

Το σύστημα διασύνδεσης Enumeration δεν είναι μια αυτοτελής δομή δεδομένων, αλλά απλά χρησιμοποιείται από τις άλλες δομές δεδομένων για την ανάκτηση διαδοχικών στοιχείων.

Βασική έννοια σε όλες τις παραπάνω κλάσεις είναι η σήμανση. Μια σήμανση (flag) είναι μια λογική τιμή, η οποία χρησιμοποιείται για την αναπαράσταση της κατάστασης on / off (ενεργό / ανενεργό) σε ένα πρόγραμμα. Ταυτόχρονα είναι καλό να καταλάβετε και την έννοια του κλειδιού. Πρόκειται ουσιαστικά για μια αριθμητική τιμή, η οποία χρησιμοποιείται για την αναζήτηση ή την αναφορά σε μια τιμή που είναι αποθηκευμένη σε μια δεδομένη βάση δεδομένων.

26. Κλάσεις

26.1 Η κλάση BitSet

Η κλάση BitSet υλοποιεί μια ομάδα δυαδικών ψηφίων ή σημάνσεων τα οποία μπορούν να λάβουν τιμές 1 και 0. Πρόκειται για μια πολύ χρήσιμη κλάση στην περίπτωση που θέλετε να αποθηκεύσετε μια ομάδα λογικών τιμών. Δύο είναι τα βασικά χαρακτηριστικά που διαφοροποιούν την κλάση αυτή : επιτρέπει τη χρησιμοποίηση μεμονωμένων bits για την αναπαράσταση των δεδομένων, αλλά και την αυτόματη αύξηση του μεγέθους της για την αναπαράσταση των bit που χρειάζονται να αποθηκευτούν.

Για τη δημιουργία ενός αντικειμένου της κλάσης BitSet απλά χρησιμοποιείτε την μέθοδο δημιουργίας :

BitSet bits = new BitSet( );

Μπορείτε να διαβάσετε τη τιμή ενός μόνο bit ενός BitSet χρησιμοποιώντας τη μέθοδο get(). Για παράδειγμα :

boolean canIWrite = bits.get (someBits.writeable);

Επιπλέον μπορείτε να προσδιορίσετε το πλήθος των bits που αντιπροσωπεύονται από ένα BitSet με τη μέθοδο size(). Για παράδειγμα:

```
int  numBits = bits.size( ) ;
```

Η κλάση αυτή παρέχει επίσης και κάποιες μεθόδους για τη πραγματοποίηση πράξεων μεταξύ των διαφόρων bit. Το μόνο αντικείμενο που δέχονται ως παράμετρο είναι το BitSet.

26.2 Η κλάση Vector

Η κλάση Vector που παρέχεται από την Java είναι η πιο χρήσιμη, βασισμένη σε πίνακες, κλάση για την αποθήκευση των δεδομένων. Πρόκειται ουσιαστικά για έναν αναπτυσσόμενο πίνακα καθώς ολοένα και περισσότερα στοιχεία προστίθενται σε αυτόν κι έτσι ο πίνακας μεγαλώνει. Φυσικά είναι δυνατόν ο πίνακας αυτός να μικραίνει.

 Η κλάση java.util.Vector δεν μεγεθύνεται ή συρρικνώνεται ακριβώς. Απλά όταν απαιτείται από κάποια λειτουργία, δημιουργείται ένας νέος πίνακας και εκεί αντιγράφονται τα δεδομένα από τον παλιό πίνακα. Με τον τρόπο αυτό δίνεται η εντύπωση ότι έχει μεταβληθεί το μέγεθος του πίνακα.

Όλες οι λειτουργίες με τους Vector μπορεί να αποβούν χρονοβόρες αν χρησιμοποιηθούν με λανθασμένο τρόπο. Εφόσον η δομή του Vector είναι παρόμοια με αυτή των στοιβών είναι καλό να χρησιμοποιούνται για την υλοποίηση των λειτουργιών των στοιβών.

Η ακόλουθη μίνι - εφαρμογή χρησιμοποιεί ένα υπόδειγμα της κλάσης Vector στην οποία αποθηκεύει τις συντεταγμένες, στις οποίες ανατρέχει το

ποντίκι. Ο λόγος που χρησιμοποιείται ένα άνυσμα και όχι ένας πίνακας είναι διότι τα ανύσματα επιτρέπουν την εύκολη προσθήκη και διαγραφή των στοιχείων του.

Ο κώδικας του παραδείγματος αυτού είναι: VectorExample

Για τη δημιουργία ενός υποδείγματος της κλάσης Vector απλά γράφετε τα παρακάτω:

Vector listOfPositions = new Vector();

Πολλές φορές επικρατεί σύγχυση γύρω από την έννοια του μεγέθους και της χωρητικότητας ενός ανύσματος (Vector). Το μέγεθος ενός ανύσματος είναι το πλήθος των στοιχείων που είναι αποθηκευμένα σε αυτό τη δεδομένη χρονική στιγμή. Η χωρητικότητα του είναι το ποσό της μνήμης που έχει δεσμευτεί για την αποθήκευση διαφόρων στοιχείων και είναι πάντα μεγαλύτερη ή ίση με το μέγεθος του. Στο παράδειγμά μας η χωρητικότητα περιορίζεται στα 50 το πολύ στοιχεία.

Υπάρχουν ορισμένες βασικές μέθοδοι για τη διαχείριση των στοιχείων που περιέχονται σε έναν Vector. Έτσι λοιπόν, για να προσθέσετε ένα νέο στοιχείο χρησιμοποιείτε τη μέθοδο addElement. Για παράδειγμα :

v.addElement ("blue");

v.addElement("green");

Επειδή όμως θέλουμε να αποθηκεύουμε τις συντεταγμένες στις οποίες ανατρέχει το ποντίκι θα καλέσουμε τη μέθοδο αυτή μέσα στη μέθοδο MouseMoved(), η οποία θα μας επιστρέφει τις αντίστοιχες συντεταγμένες.

```
public class mouseMoved
{
 public void mouseMoved( MouseEvent e )
```

```
    {

        if ( listOfPositions.size() >= 50 )
        {
        // Διέγραψε το πρώτο στοιχείου του ανύσματος
        listOfPositions.removeElementAt( 0 );
        }

        // Πρόσθεσε την νέα θέση στο τέλος της λίστα αυτής
        listOfPositions.addElement( new Point( e.getX(), e.getY() ) );

        repaint();
    }
}
```

 Η απαρίθμηση των θέσεων των στοιχείων ξεκινά από το μηδέν. Αυτό σημαίνει ότι το κορυφαίο στοιχείο σε ένα Vector έχει θέση μηδέν.

Η μέθοδος paint() διατρέχει το άνυσμα προκειμένου να πάρει τις νέες συντεταγμένες κάθε φορά και να τις σχεδιάσει στην οθόνη. Η μέθοδος elementAt() της κλάσης Vector σας επιτρέπει να χρησιμοποιήσετε ένα δείκτη θέσης (Point) στο άνυσμα που δημιουργήσατε για να ανακτάτε κάποιο στοιχείο του.

```
public class paint
{
 public void paint( Graphics g )
```

```
  {
     g.setColor( Color.white );
     for ( int j = 1; j < listOfPositions.size(); ++j )
     {
        Point A = (Point)(listOfPositions.elementAt(j-1));
        Point B = (Point)(listOfPositions.elementAt(j));
        g.drawLine( A.x, A.y, B.x, B.y );
     }
   }
}
```

Αν θέλετε να προσπελάσετε το τελευταίο στοιχείο που προσθέσατε στο Vector χρησιμοποιείστε την μέθοδο lastElement() . Για παράδειγμα :

```
String s = (String) v.lastElement( );
```

Η μετατροπή σε τύπο String είναι απαραίτητη γιατί ο τύπος που επιστρέφει η μέθοδος αυτή είναι Object.

26.3 Η κλάση HastTable

Η κλάση HastTable παράγεται από την κλάση Dictionary. Παρέχει ένα τρόπο για την οργάνωση των δεδομένων με βάση μια δομή κλειδιού που καθορίζεται από το χρήστη. Οι πίνακες αντιστοιχίσεων έχουν συγκεκριμένη αποτελεσματικότητα που ορίζεται από το συντελεστή φόρτου του πίνακα. Ο συντελεστής φόρτου είναι ένας αριθμός μεταξύ του 0.0 και του 1.0 που καθορίζει πότε και πως θα δεσμεύει ο πίνακας χώρο για περισσότερα στοιχεία.

Οι πίνακες έχουν μια χωρητικότητα που καθορίζει το ποσό της μνήμης που δεσμεύεται από τον πίνακα. Όταν ο συντελεστής φόρτου είναι κοντά στο

1.0, τότε υπάρχει πιο αποτελεσματική χρήση της μνήμης, αλλά μεγαλύτεροι χρόνοι αναζητήσεων κάποιων στοιχείων. Οι αναζητήσεις είναι πιο αποτελεσματικές αν ο συντελεστής έχει τιμή κοντά στο 0.0 αλλά σπαταλείται περισσότερη μνήμη.

Για τη δημιουργία ενός υποδείγματος της κλάσης HashTable υπάρχουν τρεις μέθοδοι :

HashTable hash = new HashTable( );

HashTable hash = new HashTable(int); Το όρισμα προσδιορίζει την χωρητικότητα.

HashTable hash = new HashTable(int, int); Το πρώτο όρισμα προσδιορίζει την χωρητικότητα και το δεύτερο τον συντελεστή φόρτου.

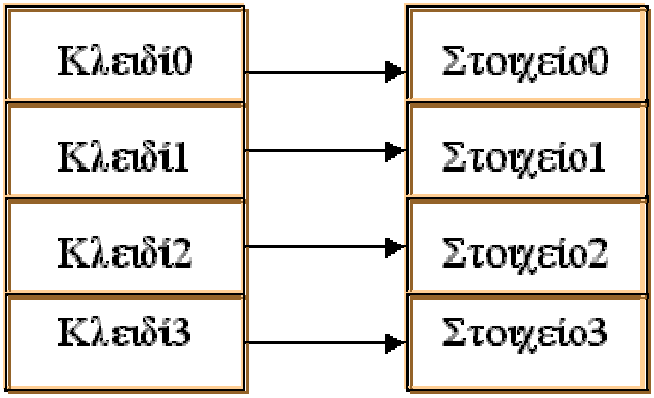
 Οι πίνακες αντιστοίχισης είναι χρήσιμοι στην περίπτωση που χειρίζεστε πολύπλοκα δεδομένα, όπου είναι πιο αποτελεσματική η προσπέλασή τους με ένα κλειδί.

26.4 Η κλάση Dictionary

Η κλάση Dictionary είναι μία αφηρημένη κλάση η οποία ορίζει το σκελετό μιας δομή δομένων, η οποία χρησιμοποιεί αντιστοίχηση μέσω κλειδιού. Η δυνατότητα αυτή είναι πολύ χρήσιμη σε περίπτωση που θέλετε να προσπελάσετε τα δεδομένα μέσω ενός κλειδιού και όχι μέσω ενός δείκτη. Εφόσον πρόκειται για μία αφηρημένη κλάση, δεν μπορείτε να δημιουργήσετε πραγματικά αντικείμενα της κλάσης. Ωστόσο όμως, μπορείτε να χρησιμοποιήσετε άλλες μεθόδους της κλάσης αυτής.

Η λειτουργία της κλάσης αυτής είναι παρόμοια με αυτή της κλάσης Vector. Προσπελάζουν τα στοιχεία μέσω στοιχείων ειδικής μορφής.

Η ακόλουθη εικόνα παρουσιάζει τον τρόπο με τον οποίο αντιστοιχίζονται τα κλειδιά (keys) στα δεδομένα.



Για να προσδιορίσετε το μέγεθος ενός αντικειμένου της κλάσης Dictionary θα πρέπει να χρησιμοποιήσετε τη μέθοδο size():

int size= dictionary.size();

Η μεταβλητή dictionary είναι ένα υπόδειγμα της κλάσης Dictionary.

Επίσης, μπορείτε να χρησιμοποιήσετε τις μεθόδους keys και elements για να πάρετε μία απαρίθμηση όλων των κλειδιών και όλων των στοιχείων αντίστοιχα που περιέχονται στο υπόδειγμα dictionary:

Enumeration keys= dictionary.keys();

Enumeration elements= dictionary.elements();

 Δεδομένου ότι όλα τα κλειδιά αντιστοιχίζονται ένα προς ένα, οι απαριθμήσεις αυτές έχουν το ίδιο μέγεθος.

27. Μονοδιάστατοι Πίνακες

Οι πίνακες είναι ο πιο αποτελεσματικός τρόπος αποθήκευσης ενός μεγάλου όγκου δεδομένων. Ένας πίνακας είναι ένα σύνολο μεταβλητών με ίδιο όνομα που είναι αποθηκευμένες σε συνεχόμενες θέσεις μνήμης οι οποίες αριθμούνται το 0 μέχρι το μέγεθος του πίνακα μείον 1. Ο αριθμός των μεταβλητών που μπορεί να αποθηκευτεί ονομάζεται διάσταση του πίνακα, ενώ κάθε μεταβλητή ονομάζεται στοιχείο του πίνακα. Ένας πίνακας μπορεί να αναπαρασταθεί σαν γραμμή ή στήλη δεδομένων όπως ο ακόλουθος πίνακας:

0	1	2	3	4
1	2	76	-90	2

ή

I[0]	1
I[1]	2
I[2]	76
I[3]	-90
I[4]	2

Ο πίνακας αυτός είναι πίνακας ακεραίων με μέγεθος 5 στοιχεία.

27.1 Δήλωση Μονοδιάστατων Πινάκων και Δέσμευση Μνήμης

Όταν δηλώνουμε ένα μονοδιάστατο πίνακα στη Java γράφουμε το όνομα του ακολουθούμενο από το σύμβολο με τις αγκύλες [] στις οποίες γράφουμε το μέγεθος του πίνακα. Ακολουθούν μερικά παραδείγματα δήλωσης μονοδιάστατων πινάκων:

```
int a[]=new int [5];

float b[]=new float [10];

String names[]=new String [10];
```

Ο τελεστής new είναι δεσμευμένη λέξη στη Java και χρησιμοποιείται για να δεσμεύσει την απαιτούμενη μνήμη για τους πίνακες.

27.2 Δήλωση και Αρχικοποίηση Μονοδιάστατων Πινάκων

Στη Java είναι εφικτό να δηλώνουμε και να αρχικοποιούμε ταυτόχρονα ένα πίνακα όπως φαίνεται στα ακόλουθα παραδείγματα:

```
int a[]={4, 5, 2};

float b[]={4.5, 6, 7.8, 9}
```

Ο πίνακας a έχει μέγεθος 3 διότι αυτό είναι το πλήθος των τιμών με τις οποίες αρχικοποιείται, ενώ ο πίνακας b έχει αντίστοιχα μέγεθος 4.

28. Δισδιάστατοι Πίνακες

Ο πιο συνηθισμένος τύπος πολυδιάστατου πίνακα είναι ο δισδιάστατος. Ένας δισδιάστατος πίνακας σαν πίνακας τιμών είναι όπως ο παρακάτω:

3	3	1	9
4	-2	3	0
5	67	65	2
45	7	2	3
87	8	0	8

Εδώ έχουμε ένα πίνακα με 5 γραμμές και 4 στήλες. Έχει συνολικά 20 στοιχεία. Έχουμε πει ότι έχει διαστάσεις 5x4 , όχι 20. Αυτός ο πίνακας δεν είναι ίδιος με τον 4x5 πίνακα που ακολουθεί πιο κάτω:

3	23	1	9	5
4	-2	3	0	6
5	67	65	2	7
45	7	2	3	9
87	8	0	8	0

Πρέπει να χρησιμοποιούμε 2 αριθμούς για να αναγνωρίσουμε τη θέση σε ένα δισδιάστατο πίνακα. Αυτές είναι οι θέσεις των γραμμών και των στηλών που βρίσκονται τα στοιχεία. Για παράδειγμα αν ο παραπάνω πίνακας λέγεται J τότε J[0][0] είναι 3, J[0][1] είναι 23, J[0][2] είναι 1, J[0][3] είναι 9, J[1][0] είναι 4 κτλ.

Οι δισδιάστατοι πίνακες δηλώνονται, ορίζονται και αρχικοποιούνται όπως και οι μονοδιάστατοι πίνακες. Γιαυτό πρέπει να καθορίζουμε 2 διαστάσεις αντί για μία και να χρησιμοποιήσουμε δύο εμφωλευμένες εντολές επανάληψης για όλες τις λειτουργίες των πινάκων.

28.1 Δήλωση Δισδιάστατων Πινάκων και Δέσμευση Μνήμης

Ένας δισδιάστατος πίνακας δηλώνεται πάντα με 2 αγκύλες στις οποίες προσδιορίζονται το πλήθος γραμμών και στηλών του πίνακα αντίστοιχα. Ακολουθούν μερικά παραδείγματα δήλωσης δισδιάστατων πινάκων:

```
int a[][]=new int [3][3];
```

```
float b[][]=new float [3][4];
```

28.2 Δήλωση και Αρχικοποίηση Δισδιάστατων Πινάκων

Στη Java είναι εφικτό να δηλώνουμε και να αρχικοποιούμε ταυτόχρονα ένα δισδιάστατο πίνακα όπως φαίνεται στα ακόλουθα παραδείγματα:

```
int a[][]={{4, 5, 2}, {5, 6, 7}};
```

```
float b[][]={4.5, 6}, {7.8, 9}, {1, 2}};
```

Ο πίνακας a έχει διάσταση 2X3 ενώ ο πίνακας b έχει διάσταση 3X2

29. Αλγόριθμοι Μονοδιάστατων Πινάκων

29.1 Διάβασμα Μονοδιάστατου Πίνακα

Για να διαβάσουμε (γεμίσουμε) ένα μονοδιάστατο πίνακα με τυχαίες τιμές γράφουμε τον ακόλουθο κώδικα:[1]

```
public class diabasma
{
static void diabasma(int x[])
{
    int i;

    for (i=0;i<5;i++)
        x[i]=(int)(Math.random()*100)+1;
}
}
```

Για να διαβάσουμε (γεμίσουμε) ένα μονοδιάστατο πίνακα εισάγοντας τιμές από το πληκτρολόγιο γράφουμε τον ακόλουθο κώδικα:[2]

```
public class diabasma2
{
```

```
static void diabasma(int x[])
{
    int i;

    for (i=0; i<x.length; i++)
    {
        System.out.println ("Δώσε "+(i+1)+" στοιχείο");
        x[i]=readint();
    }
}
}
```

Το πεδίο length περιλαμβάνει το μέγεθος ενός πίνακα.

Παρατήρηση

Η συνάρτηση *random* την οποία λαμβάνουμε μέσα από το πακέτο *Math* επιστρέφει ένα τυχαίο double αριθμό από 0 μέχρι τον προηγούμενο από αυτόν που γράφουμε στο *. Π.χ. γράφοντας *Math.random*()*100 παράγεται ένας τυχαίος αριθμός από 0 μέχρι 99. Η δήλωση (*int*) που γράφουμε μπροστά από την *random*() μετατρέπει τον double αριθμό της *rand* σε *int*. Στο αποτέλεσμα προσθέτουμε 1 για να πάρουμε ακέραιο από 1 μέχρι 100.

29.2 Εκτύπωση Μονοδιάστατου Πίνακα

Για να εκτυπώσουμε ένα μονοδιάστατο πίνακα γράφουμε τον ακόλουθο κώδικα:[3]

```
public class ektyposi
{
static void ektyposi(int x[])
{
    int i;

    for (i=0;i<5;i++)
        System.out.print(x[i]+"\t");

    System.out.println();
}
}
```

29.3 Υπολογισμός Μέσου Όρου Μονοδιάστατου Πίνακα

Για να υπολογίσουμε το μέσο όρο των στοιχείων ενός μονοδιάστατου πίνακα αθροίζουμε πρώτα όλα τα στοιχεία του πίνακα και στη συνέχεια

διαιρούμε το άθροισμα με το συνολικό πλήθος των στοιχείων του πίνακα. Αυτή η λειτουργία υλοποιείται με τον ακόλουθο κώδικα:

```
public class mesos_oros
{
static float mesos_oros(int x[])
{
        int i, sum=0;


        for (i=0;i<5;i++)
                sum+=x[i];


        return (float)sum/5;
}
}
```

29.4 Υπολογισμός πλήθους άρτιων-περιττών στοιχείων

Για να υπολογίσουμε το πλήθος άρτιων και περιττών στοιχείων όλου του πίνακα χρησιμοποιούμε 2 μετρητές, τον μετρητή art ο οποίος υπολογίζει το πλήθος των άρτιων στοιχείων και το μετρητή per ο οποίος υπολογίζει αντίστοιχα το πλήθος των περιττών στοιχείων του πίνακα. Οι 2 μετρητές πρέπει να αρχικοποιηθούν με μηδέν πριν την επανάληψη. Η λειτουργία αυτή υλοποιείται με το ακόλουθο τμήμα κώδικα:

```
public class artioi_perittoi
{
static void artioi_perittoi(int x[])
{
    int i, a=0,p=0;

    for (i=0; i<x.length; i++)
        if (x[i]%2==0)
            a++;
        else
            p++;

    System.out.println (¨Άρτιοι=¨+a¨Περιττοί=¨+p);
}
}
```

29.5 Υπολογισμός πλήθους πρώτων στοιχείων

Για να υπολογίσουμε το πλήθος των πρώτων στοιχείων του πίνακα (πρώτοι αριθμοί είναι αυτοί που διαιρούνται μόνο με τον εαυτό τους και τη μονάδα) μετράμε τους διαιρέτες του κάθε στοιχείου του πίνακα και αν το πλήθος των διαιρετών είναι <=2 τότε το στοιχείο είναι πρώτος αριθμός αλλιώς

όχι. Τους πρώτους αριθμούς τους μετράμε με ένα ξεχωριστό μετρητή ενώ τους μη πρώτους τους αγνοούμε.

Η λειτουργία αυτή υλοποιείται με το ακόλουθο τμήμα κώδικα:

```
public class protoi_arithmoi
{
static int protoi_arithmoi (int x[])
{
    int i, j, p=0, d;

    for (i=0; i<x.length; i++)
    {
        d=0;

        for (j=1;j<=x[i]; j++)
            if (x[i]%j==0)
                d++;

        if (d<=2)
            p++;
     }
```

```
    return p;
}
}
```

Παρατήρηση

Ο μετρητής d που υπολογίζει το πλήθος διαιρετών του κάθε στοιχείου του πίνακα μηδενίζεται σε κάθε εξωτερική επανάληψη η οποία «σαρώνει» τα στοιχεία του πίνακα.

29.6 Υπολογισμός πλήθους τέλειων στοιχείων πίνακα

Για να υπολογίσουμε το πλήθος των τέλειων στοιχείων του πίνακα (τέλειοι αριθμοί είναι αυτοί που το άθροισμα των διαιρετών τους ισούται με το διπλάσιο του αριθμού) αθροίζουμε τους διαιρέτες του κάθε στοιχείου του πίνακα και ελέγχουμε αν το άθροισμα αυτό είναι ίσο με το διπλάσιο του στοιχείου οπότε τότε το στοιχείο είναι τέλειος αριθμός αλλιώς όχι. Τους τέλειους αριθμούς τους μετράμε με ένα ξεχωριστό μετρητή ενώ τους μη τέλειους τους αγνοούμε.

Η λειτουργία αυτή υλοποιείται με το ακόλουθο τμήμα κώδικα:

```
public class teleioi_arithmoi
{
static int teleioi_arithmoi (int x[])
{
```

```
    int i, j, t=0, sum;


    for (i=0; i<x.length; i++)
    {
        sum=0;


        for (j=1; j<=x[i]; j++)
            if (x[i]%j==0)
                sum+=j;


        if (sum<=2*x[i])
            t++;
     }


    return t;
}
}
```

Παρατήρηση

Ο μετρητής sum που υπολογίζει το άθροισμα διαιρετών του κάθε στοιχείου του πίνακα μηδενίζεται σε κάθε εξωτερική επανάληψη η οποία «σαρώνει» τα στοιχεία του πίνακα.

29.7 Εύρεση μέγιστου-ελάχιστου στοιχείου πίνακα και των θέσεων τους σε μονοδιάστατο πίνακα

Για να βρούμε το μέγιστο και το ελάχιστο στοιχείο του πίνακα και τις θέσεις τους στον πίνακα υποθέτουμε αρχικά ότι ο μέγιστο και το ελάχιστο στοιχείο του πίνακα είναι το πρώτο (αποθηκεύουμε το μέγιστο και το ελάχιστο στοιχείο του πίνακα στις μεταβλητές max και min αντίστοιχα). Επίσης στις μεταβλητές maxp και minp αποθηκεύουμε τη θέση του μέγιστου και του ελάχιστο στοιχείο του πίνακα αντίστοιχα. Η λειτουργία αυτή υλοποιείται με το ακόλουθο τμήμα κώδικα:

```
public class max_min
{
static void max_min (int x[])
{
    int i, max, maxp, min, minp;

    max=min=x[0];
    maxp=minp=0;

    for (i=1; i<x.length; i++)
    {
        if (x[i]>max)
        {
```

```
            max=x[i];
            maxp=i;
        }

        if (x[i]<min)
        {
            min=x[i];
            minp=i;
        }
    }
    System.out.println ("Μέγιστος = "+max+" στη θέση "+(maxp+1));

    System.out.println ("Ελάχιστος = "+min+" στη θέση "+(minp+1));
}
}
```

30. Αριθμητικές Πράξεις Μονοδιάστατων Πινάκων

Στις αριθμητικές πράξεις μονοδιάστατων πινάκων παραθέτουμε ένα πλήρες πρόγραμμα για αυτές προκειμένου να τονίσουμε τις ιδιαιτερότητες της

Java όπως για παράδειγμα τη δήλωση των πινάκων και τη δέσμευση μνήμης γιαυτούς καθώς επίσης το διάβασμα και την εκτύπωση των πινάκων.

Ακολούθως παρατίθενται οι συναρτήσεις για πρόσθεση και πολλαπλασιασμό πινάκων:

```
public class athroisma
{
static void athroisma(int x[], int y[], int a[])
{
   int i;

   for (i=0; i<x.length; i++)
      a[i]=x[i] + y[i];
}

static void ginomeno(int x[], int y[], int p[])
{
   int i;

   for (i=0; i<x.length; i++)
      p[i]+=x[i] * y[i];
}}
```

Παρατήρηση

Για τους πίνακες που χρησιμοποιούνται στο πρόγραμμα είναι απαραίτητη η δέσμευση μνήμης και αυτή γίνεται στο main με την εντολή new και το μέγεθος του κάθε πίνακα.

[1] Υποθέτουμε ότι σε όλες τις λειτουργίες που αναφέρουμε ότι έχει προηγηθεί η δήλωση *int* x[5];

31. Αλγόριθμοι Δισδιάστατων Πινάκων

31.1 Διάβασμα Δισδιάστατου Πίνακα

Για να διαβάσουμε (γεμίσουμε) ένα δισδιάστατο πίνακα κατά γραμμές με τυχαίες τιμές π.χ. από 1 μέχρι 100 γράφουμε τον ακόλουθο κώδικα:[1]

```
public class diabasmsa
{
static void diabasma(int x[][])
{
    int i, j;

    for (i=0;i<3;i++)
```

```
        for (j=0;j<3;j++)
                x[i][j]=(int)(Math.random()*100)+1;
}
}
```

Για να διαβάσουμε (γεμίσουμε) ένα δισδιάστατο πίνακα κατά στήλες με τυχαίες τιμές π.χ. από 1 μέχρι 100 γράφουμε τον ακόλουθο κώδικα:

```
public class diabasma2
{
static void diabasma(int x[][])
{
    int i, j;

    for (j=0;j<3;j++)
        for (i=0;i<3;i++)
            x[i][j]=(int)(Math.random()*100)+1;
}
}
```

Για να διαβάσουμε (γεμίσουμε) ένα δισδιάστατο πίνακα κατά γραμμές εισάγοντας τιμές από το πληκτρολόγιο γράφουμε τον ακόλουθο κώδικα:

```
public class diabasma3
{
static void diabasma(int x[][])
{
    int i, j;


    for (i=0;i<3;i++)
        for (j=0;j<3;j++)
        {
System.out.println("Δώσε"+(i+1)+","+(j+1)+"στοιχείο");
            x[i][j]=readint();
        }
}
}
```

Για να διαβάσουμε (γεμίσουμε) ένα δισδιάστατο πίνακα κατά στήλες εισάγοντας τιμές από το πληκτρολόγιο γράφουμε τον ακόλουθο κώδικα:

```
public class diabasma4
{
static void diabasma(int x[][])
```

```
{
    int i, j;


    for (j=0;j<3;j++)
        for (i=0;i<3;i++)
        {
System.out.println("Δώσε"+(j+1)+",□+□(i+1)+□στοιχείο");
            x[i][j]=readint();
        }
}
}
```

Η συνάρτηση readint() είναι ακριβώς ίδια με αυτή που αναφέραμε και στους μονοδιάστατους πίνακες.

31.2 Εκτύπωση Δισδιάστατου Πίνακα

Για να εκτυπώσουμε ένα δισδιάστατο πίνακα γράφουμε τον ακόλουθο κώδικα:

```
public class ektyposi
{
```

```
static void ektyposi(int x[][])
{
    int i, j;

    for (i=0; i<3; i++)
    {
        for (j=0;j<3;j++)
            System.out.print(x[i][j]+"\t");

        System.out.println();
    }
}
}
```

31.3 Υπολογισμός Μέσου Όρου Δισδιάστατου Πίνακα

Για να υπολογίσουμε το μέσο όρο των στοιχείων ενός δισδιάστατου πίνακα με συνάρτηση η οποία τον επιστρέφει γράφουμε το ακόλουθο τμήμα κώδικα:

```
public class mesos_oros
{
static float mesos_oros(int x[][])
{
    int i, j, sum=0;

    for (i=0;i<3;i++)
        for (j=0;j<3;j++)
            sum+=x[i][j];

    return (float)sum/9;
}
}
```

Για να υπολογίσουμε το μέσο όρο των στοιχείων ενός δισδιάστατου πίνακα με συνάρτηση η οποία τον τυπώνει γράφουμε το ακόλουθο τμήμα κώδικα:

```
public class mesos_oros2
{
void float mesos_oros(int x[][])
```

```
{
    int i, j, sum=0;


    for (i=0;i<3;i++)
        for (j=0;j<3;j++)
            sum+=x[i][j];


    System.out.println(□Μέσος όρος = □+(float)sum/9);
}
}
```

31.4 Υπολογισμός Μέσου Όρου Κάθε Γραμμής Δισδιάστατου Πίνακα

Για να υπολογίσουμε το μέσο όρο των στοιχείων κάθε γραμμής ξεχωριστά ενός δισδιάστατου πίνακα γράφουμε το ακόλουθο τμήμα κώδικα:

```
public class mo_grammon
{
static void mo_grammon(int x[][])
{
    int i, j, sum;
```

```
    float mo=0;


    for (i=0;i<3;i++)
    {
        sum=0;


        for (j=0;j<3;j++)
            sum+=x[i][j];


        mo=(float)sum/3;
        System.out.println("Μέσος όρος "+(i+1)+" γραμμής="+mo);
    }
}
}
```

31.5 Υπολογισμός Μέσου Όρου Κάθε Στήλης Δισδιάστατου Πίνακα

Για να υπολογίσουμε το μέσο όρο των στοιχείων κάθε στήλης ξεχωριστά ενός δισδιάστατου πίνακα γράφουμε το ακόλουθο τμήμα κώδικα:

```
public class mo_stilon
{
```

```
static void mo_stilon(int x[][])
{
    int i, j, sum;
    float mo=0;

    for (j=0;j<3;j++)
    {
        sum=0;

        for (i=0;i<3;i++)
            sum+=x[i][j];

        mo=(float)sum/3;
        System.out.println ("Μέσος όρος "+(j+1)+" στήλης ="+mo);
    }
}
}
```

31.6 Υπολογισμός Μέγιστου-Ελάχιστου στοιχείου Δισδιάστατου Πίνακα και των θέσεων τους στον πίνακα

Για να υπολογίσουμε το μέγιστο και το ελάχιστο στοιχείο ενός δισδιάστατου πίνακα καθώς και τις θέσεις τους στον πίνακα με συνάρτηση η οποία τυπώνει όλα τα αποτελέσματα γράφουμε το ακόλουθο τμήμα κώδικα (στο πρόγραμμα αυτό τυπώνουμε όλες τις θέσεις του δισδιάστατου πίνακα στις οποίες συναντούμε το μέγιστο και το ελάχιστο στοιχείο του πίνακα):

```
public class max_min
{
static void max_min (int x[][])
{
    int i, j, max, maxp, min, minp, maxpi, minpi, maxpj, minpj;

    max=min=x[0][0];
    maxpi=minpi=maxpj=minpj=0;

    for (i=1;i<3;i++)
        for (j=0;j<3;j++)
        {
            if (x[i][j]>max)
            {
                max=x[i][j];
```

```
            maxpi=i;
            maxpj=j;
        }

        if (x[i][j]<min)
        {
            min=x[i][j];
            minpi=i;
            minpj=j;
        }
    }
    System.out.println ("Μέγιστο = "+max);
    for (i=1;i<3;i++)
        for (j=0;j<3;j++)
            if (x[i][j]==max)
                System.out.println("θέση"+(i+1)+","+(j+1));
    System.out.println("Ελάχιστο= "+min);
    for (i=1;i<3;i++)
        for (j=0;j<3;j++)
            if (x[i][j]==min)
```

```
                                    System.out.println("θέση"+(i+1)+","+(j+1))}
}
```

Για να υπολογίσουμε το μέγιστο και το ελάχιστο στοιχείο ενός δισδιάστατου πίνακα καθώς και τις θέσεις τους στον πίνακα με συνάρτηση η οποία επιστρέφει όλα τα αποτελέσματα γράφουμε το ακόλουθο τμήμα κώδικα:

```
public class max_min2
{
static String max_min (int x[][])
{
    int i, j, max, maxp, min, minp, maxpi, minpi, maxpj, minpj;
    String res=□    ;□


    max=min=x[0][0];
    maxpi=minpi=maxpj=minpj=0;


    for (i=1;i<3;i++)
        for (j=0;j<3;j++)
        {
            if (x[i][j]>max)
            {
```

```
            max=x[i][j];
            maxpi=i;
            maxpj=j;
        }

        if (x[i][j]<min)
        {
            min=x[i][j];
            minpi=i;
            minpj=j;
        }
    }

    res+="Μέγιστο = "+max;
    for (i=1;i<3;i++)
        for (j=0;j<3;j++)
            if (x[i][j]==max)
                res+="θέση"+(i+1)+ ","+(j+1));

    res+="\nΕλάχιστο= "+min);
    for (i=1;i<3;i++)
```

```
                for (j=0;j<3;j++)
                        if (x[i][j]==min)
                            res+="θέση"+(i+1)+ ","+(j+1));


        return res;
}
}
```

Παρατήρηση

Το χαρακτηριστικό του κώδικα που αναφέραμε είναι ότι συνενώνουμε όλα τα αποτελέσματα στη μεταβλητή res τύπου String και επιστρέφουμε ως αποτέλεσμα της συνάρτησης τη μεταβλητή αυτή. Ο τελεστής + όταν χρησιμοποιείται με αλφαριθμητικά στη Java κάνει συνένωση (concatenation). Ο λόγος που γίνεται αυτό είναι ότι στη Java όπως και στη C++ όλες οι συναρτήσεις μπορούν να επιστρέφουν το πολύ ένα αποτέλεσμα στη συνάρτηση από την οποία καλούνται (είτε αυτή είναι το main είτε οποιαδήποτε άλλη συνάρτηση), οπότε συνενώνοντας όλα τα αποτελέσματα σε μια μεταβλητή τύπου String τότε επιστρέφουμε πολλά αποτελέσματα από τη συνάρτηση αλλά όλα αυτά είναι τοποθετημένα σε ένα αλφαριθμητικό (συμβολοσειρά).

31.7 Πλήθος Άρτιων και Περιττών στοιχείων Δισδιάστατου Πίνακα

Ο υπολογισμός του πλήθους άρτιων και περιττών στοιχείων ενός δισδιάστατου πίνακα υλοποιείται με το ακόλουθο τμήμα κώδικα:

```
public class artioi_perittoi
{
static void artioi_perittoi (int x[][])
{
    int i, j, a=0,p=0;

    for (i=0;i<3;i++)
        for (j=0;j<3;j++)
            if (x[i][j]%2==0)
                a++;
            else
                p++;

    System.out.println(¨Πλήθος Άρτιων=¨+a¨Πλήθος Περιττών=¨+p);
}
}
```

31.8 Πλήθος Άρτιων-Περιττών στοιχείων σε κάθε γραμμή ενός Δισδιάστατου Πίνακα

Ο υπολογισμός του πλήθους άρτιων και περιττών στοιχείων σε κάθε γραμμή ενός δισδιάστατου πίνακα υλοποιείται με το ακόλουθο τμήμα κώδικα:

```
public class artioi_perittoi_grammon
{
static void artioi_perittoi_grammon(int x[][])
{
    int i, j, a, p;

    for (i=0; i<3; i++)
    {
        a=0;
        p=0;

        for (j=0; j<3; j++)
            if (x[i][j]%2==0)
              a++;
            else
              p++;
```

```
        System.out.println("Πλήθος Άρτιων "+(i+1)+"γραμμής = "+a+" Πλήθος Περιττών "+(i+1)+" γραμμής = "+p);
    }
}
}
```

31.9 Πλήθος Άρτιων-Περιττών στοιχείων σε κάθε στήλη ενός Δισδιάστατου Πίνακα

Ο υπολογισμός του πλήθους άρτιων και περιττών στοιχείων σε κάθε στήλη ενός δισδιάστατου πίνακα υλοποιείται με το ακόλουθο τμήμα κώδικα:

```
public class artioi_perittoi_stilon
{
static void artioi_perittoi_stilon(int x[][])
{
    int i, j, a, p;

    for (j=0; j<3; j++)
    {
```

```
        a=0;
        p=0;

        for (i=0; i<3; i++)
            if (x[i][j]%2==0)
              a++;
            else
              p++;

        System.out.println("Πλήθος Άρτιων "+(j+1)+" στήλης = "+a+" Πλήθος Περιττών "+(j+1)+" στήλης = "+p);
    }
}
}
```

31.10 Σειριακή Αναζήτηση Στοιχείου σε Δισδιάστατο Πίνακα

Η ακόλουθη συνάρτηση διαβάζει ένα στοιχείο αναζήτησης από το πληκτρολόγιο και τυπώνει όλες τις θέσεις στις οποίες αυτό εμφανίζεται καθώς και το πλήθος εμφανίσεων του στον πίνακα. Τα αποτελέσματα τυπώνονται μέσα στη συνάρτηση.

```
public claas search
{
static void search(int x[][])
{
    int i, j, y, t=0;

    System.out.println("Δώστε στοιχείο αναζήτησης ");
    y=readint();

    for (i=0;i<3;i++)
        for (j=0;j<3;j++)
            if (y==x[i][j])
            {
                System.out.println("Θέση"+(i+1)+","+(j+1));
                t++;
            }

    if (t==0)
        System.out.println("Το στοιχείο δεν υπάρχει");
    else
        System.out.println("Το στοιχείο βρέθηκε "+t+" φορές ");
```

```
}

}
```

Μια παραλλαγή της προηγούμενης αναζήτησης είναι η συνάρτηση να επιστρέφει τα αποτελέσματα της (δηλαδή τις θέσεις στις οποίες το στοιχείο υπάρχει στον πίνακα καθώς και το πλήθος των εμφανίσεων του στον πίνακα) συνενωμένα σε ένα αλφαριθμητικό στο main και αυτό να τυπώνεται στο main.

Αυτή η παραλλαγή υλοποιείται με τον ακόλουθο κώδικα:

```
public class search2
{
static String search(int x[][])
{
    int i, j, y, t=0;
    String res="";

    System.out.println("Δώστε στοιχείο αναζήτησης ");
    y=readint();

    for (i=0;i<3;i++)
        for (j=0;j<3;j++)
            if (y==x[i][j])
```

```
                {
                    res+="Θέση"+(i+1)+","+(j+1);
                    t++;
                }

    if (t==0)
        res+="Το στοιχείο δεν υπάρχει";
    else
        res+="Το στοιχείο βρέθηκε "+t+" φορές ";

    return res;
}
}
```

31.11 Άθροισμα Διαγωνίων Δισδιάστατου Πίνακα

Με την προϋπόθεση ότι ο πίνακας στον οποίο αναφερόμαστε είναι τετραγωνικός μπορούμε να υπολογίσουμε το άθροισμα της κύριας και της δευτερεύουσας διαγωνίου του με δύο τρόπους. Ο 1ος τρόπος είναι με την ακόλουθη συνάρτηση στην οποία εκτυπώνονται και τα δύο αθροίσματα:

```
public class athroisma_diagonion
{
static void athroisma_diagonion (int x[][])
{
    int i, j, sum1=0, sum2=0;


    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
        {
            if (i==j)
                sum1+=x[i][j];


            if (i+j==3-1)
                sum2+=x[i][j];
        }
    System.out.println ("Άθροισμα Κύριας Διαγωνίου = "+sum1);
    System.out.println ("Άθροισμα Δευτερεύουσας Διαγωνίου = "+sum2);
}
}
```

Ο 2ος τρόπος είναι με την ακόλουθη συνάρτηση στην οποία επιστρέφονται τα δύο αθροίσματα στο main συνενωμένα σε μια συμβολοσειρά (αλφαριθμητικό):

```
public class athroisma_diagonion2
{
static String athroisma_diagonion (int x[][])
{
    int i, j, sum1=0, sum2=0;
    String res="";

    for (i=0; i<3; i++)
        for (j=0; j<3; j++)
        {
            if (i == j)
                sum1+=x[i][j];

            if (i+j == 3-1)
                sum2+=x[i][j];
        }
```

```
    res+="Άθροισμα Κύριας Διαγωνίου = "+sum1;
    res+="Άθροισμα Δευτερεύουσας Διαγωνίου = "+sum2;


    return res;
}
}
```

31.12 Εναλλαγή Γραμμών - Στηλών Δισδιάστατου Πίνακα

Οι ακόλουθοι αλγόριθμοι παρουσιάζουν εναλλαγές γραμμών και στηλών σε ένα δισδιάστατο πίνακα.

Α περίπτωση: Εναλλαγή 1ης-Τελευταίας στήλης

```
public class enallagi_protis_kai_teleytaias_stilis
{
static void enallagi_protis_kai_teleytaias_stilis (int x[][])
{
    int i, temp;


    for (i=0;i<3;i++)
    {
        temp=x[i][0];
```

```
            x[i][0]=x[i][3-1];
            x[i][3-1]=temp;
        }
}
}
```

Β περίπτωση: Εναλλαγή 1ης-Τελευταίας σειράς (γραμμής)

```
Public class enallagi_protis_kai_teleytaias_grammis
{
static void enallagi_protis_kai_teleytaias_grammis (int x[][])
{
    int j, temp;

    for (j=0;j<3;j++)
    {
        temp=x[0][j];
        x[0][j]=x[3-1][j];
        x[3-1][j]=temp;
    }
}
```

```
}
```

[1] Υποθέτουμε ότι σε όλες τις λειτουργίες που αναφέρουμε ότι έχει προηγηθεί η δήλωση *int* x[3][3];

32. Στοίβα (Stack)

Η στοίβα περιέχει μια ακολουθία στοιχείων, τα οποία πρέπει να θεωρηθούν στοιβαγμένα το ένα πάνω στο άλλο, σαν μια φυσική στοίβα κουτιών ή δίσκων. Μόνο το στοιχείο που βρίσκεται στην κορυφή της στοίβας είναι προσπελάσιμο κάθε φορά. Μπορεί να μετακινηθεί από την στοίβα με μια μέθοδο που καλείται pop. Ένα στοιχείο κάτω στη στοίβα μπορεί να μετακινηθεί μόνο αφού έχουν μετακινηθεί όλα τα στοιχεία που βρίσκονται πάνω από αυτό στην στοίβα. Ένα νέο στοιχείο μπορεί να προστεθεί στην κορυφή της στοίβας με τη διαδικασία που καλείται push. Μπορεί να δημιουργηθεί στοίβα από οτιδήποτε αντικείμενα. Αν, για παράδειγμα, τα στοιχεία είναι τιμές τύπου int τότε οι διεργασίες pop και push της κλάσης Stack μπορούν να εφαρμοστούν ως υποδείγματα των μεθόδων :

void push (int newItem)

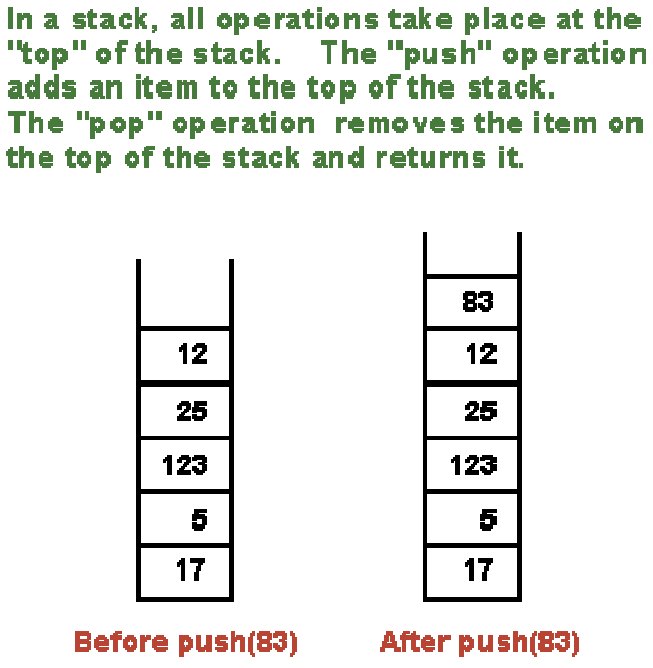
int pop( )

Είναι λάθος να προσπαθείτε να βγάλετε ένα στοιχείο από μια κενή λίστα, γι' αυτό πρέπει να είστε ικανοί να καταλάβετε πότε μια λίστα είναι κενή. Αυτό

για να γίνει χρειάζεται μια άλλη λειτουργία, που θα εφαρμόσετε σαν αντικείμενο της μεθόδου

boolean isEmpty( )

Αυτό περιγράφει μια στοίβα ακεραίων ως ένα αφηρημένο τύπο δεδομένων. Αυτός ο τύπος μπορεί να εφαρμοστεί με πολλούς τρόπους, ωστόσο η συμπεριφορά του πρέπει να ανταποκρίνεται στην αφηρημένη εικόνα της στοίβας.



Στη διασυνδεδεμένη εφαρμογή της στοίβας, η κορυφή της στοίβας είναι ο κόμβος που είναι στην κορυφή της στοίβας. Είναι εύκολο να προσθέσετε και

να μετακινήσετε κόμβους από την αρχή της διασυνδεδεμένης λίστας. Η κλάση που εφαρμόζει τον αφηρημένο τύπο δεδομένων με τη χρήση συνδεδεμένης λίστας είναι :

```
public class StackOfInts {
private static class Node {
 int item;
Node next;
 }
 private Node top;
 public void push( int N ) {
 Node newTop;
 newTop = new Node();
newTop.item = N;
newTop.next = top;
 top = newTop;
}
 public int pop() {
 int topItem = top.item;
 top = top.next;
 return topItem;
 }
public boolean isEmpty() {
return (top == null);
 } } // end class StackOfInts
```

Πρέπει να είστε σίγουροι ότι έχετε καταλάβει τον τρόπο λειτουργίας των pop και push. Η σχεδίαση ορισμένων εικόνων θα βοηθούσε. Η συνδεδεμένη λίστα είναι μέρος της εφαρμογής της κλάσης StackOfInts.

Τώρα, είναι εύκολο να υλοποιήσετε τη στοίβα ως ένα πίνακα αντί για μια συνδεδεμένη λίστα. Εφόσον ο αριθμός των στοιχείων του πίνακα ποικίλλει, ένας μετρητής χρειάζεται για να ελέγχει τον αριθμό των θέσεων του πίνακα που χρειάζονται κάθε φορά. Αν ο μετρητής αυτός καλείται top, τότε τα στοιχεία είναι αποθηκευμένα στις θέσεις 0, 1, ... top-1. Το στοιχείο στη θέση 0 είναι στο τέλος της στοίβας. Η τοποθέτηση ενός στοιχείου στη στοίβα είναι εύκολη: Απλά το προστέθετε στη θέση top και αυξάνεται κατά ένα η τιμή του top. Μια δεύτερη εφαρμογή της StackOfInts με τη χρήση δυναμικών πινάκων είναι :

```
public class StackOfInts {
private int[] items = new int[10];
private int top = 0;
 public void push( int N ) {
 if (top == items.length) {
  int[] newArray = new int[ 2*items.length ];
System.arraycopy(items, 0, newArray, 0, items.length);
items = newArray;
}
 items[top] = N;
 top++;
 }
 public int pop() {
```

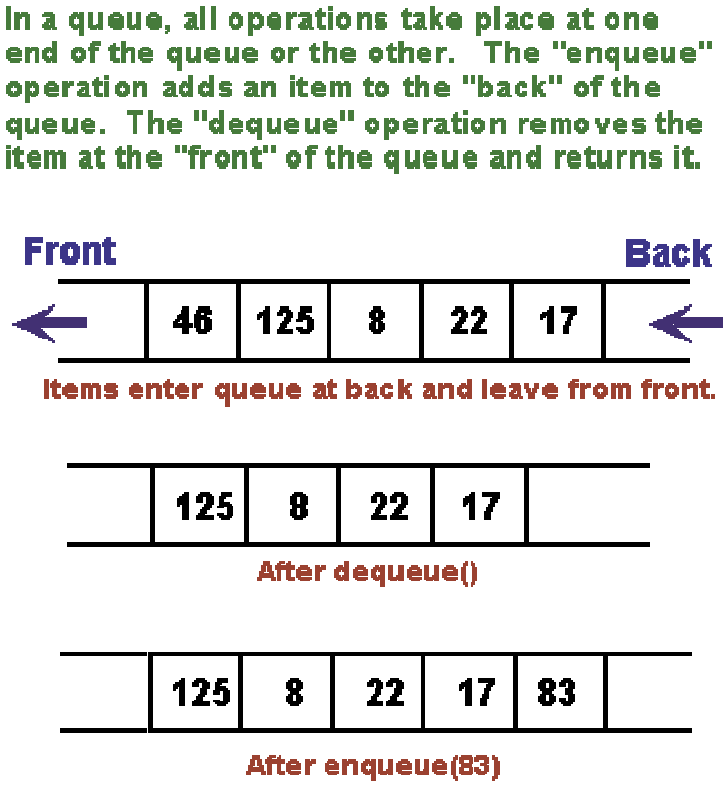
```
 int topItem = items[top - 1]
 top--;
 return topItem;
}
 public boolean isEmpty() {
return (top == 0); } } // end class StackOfInts
```

Μια ακόμη φορά, η υλοποίηση της στοίβας ως πίνακα είναι ιδιωτική στην κλάση. Οι δύο εκδοχές της κλάσης StackOfInts, μπορούν να χρησιμοποιηθούν εναλλακτικά. Αν το πρόγραμμα χρησιμοποιεί την μια εκδοχή, είναι πιθανό να αντικαταστήσετε την άλλη εκδοχή χωρίς να αλλάξετε το πρόγραμμα. Δυστυχώς, υπάρχει μια λεπτομέρεια στην οποία οι δύο κλάσεις λειτουργούν διαφορετικά: όταν γίνεται μια προσπάθεια για την ανάκτηση ενός στοιχείου από μια κενή στοίβα, η πρώτη εκδοχή δημιουργεί μια εξαίρεση NullPointerException, ενώ η δεύτερη μια εξαίρεση ArrayIndexOutOfBoundsException.

33. Ουρές (Queues)

Οι ουρές είναι όμοιες με τις στοίβες στο ότι αποτελούνται από ακολουθιακά στοιχεία, και υπάρχουν περιορισμοί στον τρόπο προσθήκης και μετακίνησης των στοιχείων από τη λίστα. Ωστόσο, μια ουρά έχει δύο άκρα, που καλούνται αρχή και τέλος της ουράς. Τα στοιχεία προστίθενται πάντα στο τέλος της ουράς και αφαιρούνται από την αρχή της ουράς. Οι λειτουργίες της προσθήκης και της μετακίνησης στοιχείων καλούνται enqueue(εισαγωγή στην ουρά) και dequeue (απομάκρυνση από την ουρά). Ένα στοιχείο που προστίθεται στο τέλος της ουράς, θα παραμείνει στην ουρά μέχρις ότου όλα τα

στοιχεία που βρίσκονται μπροστά από αυτό μετακινηθούν. Μια ουρά είναι σαν μια "σειρά" ή μια "ουρά" πελατών που περιμένουν σε μια υπηρεσία. Οι πελάτες εξυπηρετούνται ανάλογα με την σειρά τους στην ουρά.



Μια ουρά μπορεί να κρατά στοιχεία οποιουδήποτε τύπου. Για μια ουρά ακεραίων, οι δύο προηγούμενες διαδικασίες μπορούν να εφαρμοστούν ως μέθοδοι της κλάσης "QueueOfInts". Χρειάζεται επίσης και μια μέθοδος για να ελεγχθεί αν η ουρά είναι κενή.

void enqueue(int N)

int dequeue()

boolean isEmpty()

Μια ουρά μπορεί να υλοποιηθεί ως μια διασυνδεδεμένη λίστα ή ως ένας πίνακας. Μια αποδοτική υλοποίηση με πίνακα είναι λίγο πιο δύσκολη από ότι η υλοποίηση εφαρμογή της στοίβας. Στην εφαρμογή της διασυνδεδεμένης λίστας, το πρώτο στοιχείο της λίστας βρίσκεται στην αρχή της στοίβας. Η υλοποίηση ενός στοιχείου από την αρχή της ουράς είναι όπως ακριβώς η απομάκρυνση ενός στοιχείου από μια στοίβα. Το τέλος της ουράς βρίσκεται στο τέλος της λίστας. Η προσθήκη ενός στοιχείου στην ουρά, περιέχει τον καθορισμό ενός δείκτη στον τελευταίο κόμβο της τρέχουσας στοίβας για να δείχνει στο νέο κόμβο που περιέχει το στοιχείο. Για αν γίνει αυτό, χρειάζεται μία εντολή της μορφής tail.text= newNode, όπου το tail είναι ένας δείκτης στον τελευταίο κόμβο της λίστας. Αν η μεταβλητή head είναι δείκτης στον πρώτο κόμβο της λίστας, θα είναι πάντα δυνατό να πάρετε ένα δείκτη του τελευταίου στοιχείο με τον παρακάτω κώδικα:

```
Node tail;
tail = head;
while (tail.next != null) {
   tail = tail.next;
}
```

Ωστόσο, θα είναι αναποτελεσματικό να επαναλαμβάνετε αυτή τη διαδικασία κάθε φορά που ένα στοιχείο προστίθεται στην ουρά. Για λόγους αποδοτικότητας, θα υπάρχει ένας δείκτης στον τελευταίο κόμβο της μεταβλητής. Απλά πρέπει να προσέχετε και να ενημερώνετε την μεταβλητή για τις αλλαγές των τιμών. Επομένως η κλάση QueueOfInts θα είναι :

```
public class QueueOfInts {

  private static class Node {
            int item;
    Node next;
  }

  private Node head = null;
  private Node tail = null;

  void enqueue( int N ) {

    Node newTail = new Node();
    newTail.item = N;
    if (head == null) {

      head = newTail;
      tail = newTail;
    }
    else {

      tail.next = newTail;
      tail = newTail;
    }
  }

  int dequeue() {

    head = head.next;
```

```
    if (head == null) {

      tail = null;
    }
    return firstItem;
  }

  boolean isEmpty() {

    return (head == null);
  }

}
```

Οι ουρές χρησιμοποιούνται στους υπολογιστές όταν μόνο ένα στοιχείο επεξεργάζεται κάθε φορά, αλλά πολλά στοιχεία μπορούν να περιμένουν για να επεξεργαστούν. Για παράδειγμα:

Σε ένα πρόγραμμα της Java που υπάρχουν πολλά νήματα, τα νήματα που χρειάζονται επεξεργασία περιμένουν σε μια ουρά. Όταν ένα νέο νήμα ξεκινά, προστίθεται στο τέλος της ουράς. Ένα νήμα μετακινείται από την αρχή της ουράς, επεξεργάζεται και μετά αν δεν τερματιστεί επιστρέφεται στην ουρά.

Συμβάντα όπως κλικ του ποντικιού και χρησιμοποίηση των πλήκτρων, αποθηκεύονται σε μια ουρά που καλείται "ουρά συμβάντων". Το πρόγραμμα μετακινεί τα συμβάντα από την ουρά και τα επεξεργάζεται. Είναι πιθανό να εμφανιστούν πολλά άλλασυμβάντα κατά την επεξεργασία ενός συμβάντος,

αλλά εφόσον αυτά αποθηκεύονται σε μια ουρά, πάντα επεξεργάζονται με τη σειρά που εμφανίστηκαν.

Οι ουρές εφαρμόζουν την πολιτική FIFO: First In, First Out. Οι στοίβες εφαρμόζουν την πολιτική LIFO: Last In, First Out. Το στοιχείο που έρχεται έξω από τη στοίβα είναι το τελευταίο. Όπως οι ουρές, οι στοίβες μπορούν να χρησιμοποιηθούν για να κρατούν στοιχεία που περιμένουν να επεξεργαστούν. Για να καταλάβετε τις διαφορές μεταξύ των ουρών και των στοιβών απλά δείτε την παρακάτω μίνι εφαρμογή. Όταν κάνετε κλικ στο λευκό τετράγωνο του πλέγματος, η μίνι εφαρμογή θα μαρκάρει σταδιακά όλα τα τετράγωνα του πλέγματος ξεκινώντας από αυτό που κάνατε κλικ. Για να καταλάβετε πως το κάνει αυτό η μίνι εφαρμογή σκεφτείτε τον εαυτό σας στη θέση της μίνι εφαρμογής. Όταν ο χρήστης κάνει κλικ στο τετράγωνο, κρατάτε μια κάρτα. Η θέση του τετραγώνου καταγράφεται σε αυτή την κάρτα. Τοποθετείτε την κάρτα σε έναν σωρό, που μετά περιέχει μόνο μια κάρτα. Μετά επαναλαμβάνετε τα ακόλουθα: Αν ο σωρός είναι κενός τότε τελειώσατε. Διαφορετικά πρέπει να πάρετε μια κάρτα από τον σωρό. Η κάρτα προσδιορίζει το τετράγωνο. Μετά δείτε τα γειτονικά τετράγωνα αυτού του τετραγώνου. Αν δεν έχουν χρησιμοποιηθεί καταγράψτε τη θέση τους στην κάρτα.

Καθώς το τετράγωνο είναι στο σωρό και περιμένει να επεξεργαστεί χρωματίζεται κόκκινο. Όταν ένα τετράγωνο μετακινείται από το σωρό για να επεξεργαστεί, το χρώμα του αλλάζει στο γκρι. Τελικά, όλα τα τετράγωνα θα επεξεργαστούν και η διαδικασία θα τελειώσει. Στην αναλογία των καρτών, ο σωρός των καρτών περιέχει όλες τα κόκκινα τετράγωνα.

Η μίνι εφαρμογή μπορεί να χρησιμοποιήσει μια από τις τρεις επιλογές: Στοίβα, Ουρά και Τυχαία Τοποθέτηση. Σε κάθε περίπτωση, χρησιμοποιείται η ίδια διαδικασία. Η μόνη διαφορά είναι ο τρόπος χειρισμού των καρτών. Στη στοίβα, οι κάρτες προστίθενται και μετακινούνται από την αρχή του σωρού.

Στην ουρά, οι κάρτες προστίθενται στο τέλος και μετακινούνται από την αρχή. Στην τυχαία περίπτωση, η κάρτα που θα επεξεργαστεί επιλέγεται τυχαία από τον σωρό των καρτών. Η σειρά επεξεργασίας είναι πολύ διαφορετική στις τρεις περιπτώσεις.

Ο πηγαίος κώδικας για την μίνι εφαρμογή είναι DephtBreath

Οι ουρές φαίνονται πολύ φυσιολογικές επειδή εμφανίζονται πολλές φορές, αλλά υπάρχουν περιπτώσεις που οι στοίβες είναι πιο κατάλληλες. Για παράδειγμα, σκεφτείτε τι συμβαίνει όταν μια ρουτίνα καλεί μία υπό-ρουτίνα. Η πρώτη ρουτίνα διακόπτεται ενώ εκτελείται η υπό-ρουτίνα και θα συνεχίσει την εκτέλεσή της μετά τον τερματισμό της εκτέλεσης της υπό-ρουτίνας. Τώρα, υποθέστε ότι η υπό-ρουτίνα καλεί μια άλλη υπό-ρουτίνα και αυτή με τη σειρά της μια άλλη. Κάθε μια από τις υπό-ρουτίνες διακόπτεται καθώς εκτελείται η επόμενη. Ο υπολογιστής διατηρεί όλες τις υπό-ρουτίνες που διακόπτονται. Αυτό το κάνει με την χρησιμοποίηση μιας στοίβας.

Όταν καλείται μια υπό-ρουτίνα, μια εγγραφή ενεργοποίησης δημιουργείται για την υπό-ρουτίνα. Η εγγραφή αυτή περιέχει πληροφορίες σχετικές με την εκτέλεση της υπό-ρουτίνας, όπως τις μεταβλητές της και τις παραμέτρους της. Η εγγραφή ενεργοποίησης για την υπό-ρουτίνα τοποθετείται στη στοίβα. Θα μετακινηθεί από τη στοίβα και θα καταστραφεί όταν επιστρέψει η υπό-ρουτίνα. Αν αυτή καλεί άλλη υπό-ρουτίνα, η εγγραφή ενεργοποίησης της δεύτερης υπό-ρουτίνας τοποθετείται στη στοίβα, πάνω από την αντίστοιχη της πρώτης υπό-ρουτίνας. Η στοίβα θα συνεχίζει να μεγαλώνει καθώς καλούνται υπό-ρουτίνες.

Ακόμη, οι στοίβες μπορούν να χρησιμοποιηθούν για την εκτίμηση "postfix expressions". Μια κανονική μαθηματική έκφραση όπως 2 + (15 - 12) *17 καλείται infin expression. Σε μια τέτοια περίπτωση ο τελεστής βρίσκεται

μεταξύ δύο αριθμών. Στην άλλη περίπτωση ο τελεστής βρίσκεται μετά από τους δύο αριθμούς, όπως 2 2 +.

Τώρα, ας υποθέσουμε ότι θέλουμε να επεξεργαστούμε την έκφραση "2 15 12 -17 * + ", από τα αριστερά προς τα δεξιά. Το πρώτο στοιχείο που συναντήσαμε είναι το 2, αλλά τι μπορούμε να κάνουμε με αυτό ; Μέχρι στιγμής τίποτα γιατί δεν γνωρίζουμε τον τελεστή. Πρέπει να θυμόμαστε το 2 για αργότερα. Αυτό γίνεται με την πρόσθεση του σε μια στοίβα. Αν δούμε το δεύτερο στοιχείο είναι το 15 και προστίθεται και αυτό στη στοίβα. Το ίδιο συμβαίνει και με το επόμενο στοιχείο που είναι το 12. Το επόμενο στοιχείο είναι ο τελεστής - . Ο τελεστής αυτός εφαρμόζεται στους δύο αριθμούς που βρίσκονται πριν από αυτό. Επομένως η απάντηση είναι 3. Ο αριθμός αυτός θα περιμένει κι έτσι θα τοποθετηθεί στη στοίβα. Το επόμενο στοιχείο είναι το 17 και επομένως θα τοποθετηθεί στην στοίβα. Για να επεξεργαστούμε το επόμενο στοιχείο που είναι ο τελεστής * χρειάζεται η ανάκτηση των αριθμών 3 και 17 από τη στοίβα. Το αποτέλεσμα της πράξης είναι 51 και τοποθετείται στη στοίβα. Το επόμενο στοιχείο είναι ο τελεστής +. Έτσι λοιπόν, ανακτούνται οι αριθμοί 51 και 2 προστίθενται και το αποτέλεσμα 53 τοποθετείται στην στοίβα. Έτσι ολοκληρώθηκε ο υπολογισμός της έκφρασης.

Παρόλο που για τους ανθρώπους είναι πιο εύκολο να εργάζονται με infix εκφράσεις, οι profix εκφράσεις έχουν πολλά πλεονεκτήματα. Δεν απαιτούν παρενθέσεις ή άλλους κανόνες. Η σειρά με την οποία εφαρμόζονται οι τελεστές καθορίζεται από τη σειρά με την οποία εμφανίζονται στη έκφραση. Επομένως ο αλγόριθμος υπολογισμού profix εκφράσεων είναι :

```
Public class readAndEvaluate
{
static void readAndEvaluate() {
```

```
NumberStack stack;
stack = new NumberStack();
TextIO.putln();
while (TextIO.peek() != '\n') {
if ( Character.isDigit(TextIO.peek()) ) {
num = TextIO.getDouble();
stack.push(num);
TextIO.putln(" Τοποθέτηση σταθεράς " + num);
}
else {
op = TextIO.getChar();
if (op != '+' && op != '-' && op != '*' && op != '/' && op != '^') {
TextIO.putln("\n μη νόμιμος τελεστής: " + op);
return;
}
if (stack.isEmpty()) {
TextIO.putln( " Η στοίβα βρέθηκε άδεια κατά την διάρκεια της εκτίμησης " +
op);
TextIO.putln("\nΔεν υπάρχουν αρκετοί αριθμοί στην έκφραση!");
return;
}
y = stack.pop();
if (stack.isEmpty()) {
TextIO.putln( " Η στοίβα βρέθηκε άδεια κατά την διάρκεια της εκτίμησης " +
op);
TextIO.putln("\n Δεν υπάρχουν αρκετοί αριθμοί στην έκφραση!");
```

```
 return;
 }
 x = stack.pop();
 switch (op) {
case '+': answer = x + y;
 break;
case '-': answer = x - y;
 break;
 case '*': answer = x * y;
break;
case '/': answer = x / y;
break;
default: answer = Math.pow(x,y);
 }
stack.push(answer);
TextIO.putln(" Εκτιμήθηκε " + op + " και τοποθετήθηκε " + answer);
}
 skipSpaces();
if (stack.isEmpty()) {
 TextIO.putln("Δεν δόθηκε έκφραση.");
return;
}
double value = stack.pop();
TextIO.putln(" Απομακρίνθηκε το " + value + " στο τέλος της έκφρασης.");
 if (stack.isEmpty() == false) {
 TextIO.putln(" Η στοίβα δεν είναι άδεια.");
```

```
TextIO.putln("\n Δεν υπαρχουν αρκετοί τελεστές για όλους τους αριθούς!");
 return;
 }
 TextIO.putln("\nValue = " + value);
}
}
```

Τα λάθη μπορούν πολύ εύκολα να εντοπιστούν.

34. Γραμμικές Λίστες

Το πεπερασμένο σύνολο που αποτελείται από n ≥ 0 κόμβους, X[1],X[2],X[3]..,X[n] όπου ο κόμβος X[κ] προηγείται του κόμβου X[κ+1] για 1 < κ < n και ο πρώτος κόμβος είναι το στοιχείο X[1] ονομάζεται **γραμμική λίστα**. Σε όλες τις γραμμικές λίστες το πρώτο στοιχείο ονομάζεται **κεφαλή** ενώ το τελευταίο ονομάζεται **ουρά**.

Οι γραμμικές λίστες έχουν δυο βασικές κατηγορίες ως προς τη δομή τους : τις **σειριακές** και τις **συνδεδεμένες** γραμμικές λίστες. Στη δομή των σειριακών γραμμικών λιστών οι κόμβοι καταλαμβάνουν συνεχόμενες θέσεις στην κυρία μνήμη ενώ στις συνδεδεμένες γραμμικές λίστες οι κόμβοι δεν βρίσκονται κατ'ανάγκη σε συνεχόμενες θέσεις μνήμης. Μπορεί να βρίσκονται σε απομακρυσμένες θέσεις και να είναι συνδεδεμένοι μεταξύ τους με δείκτες.

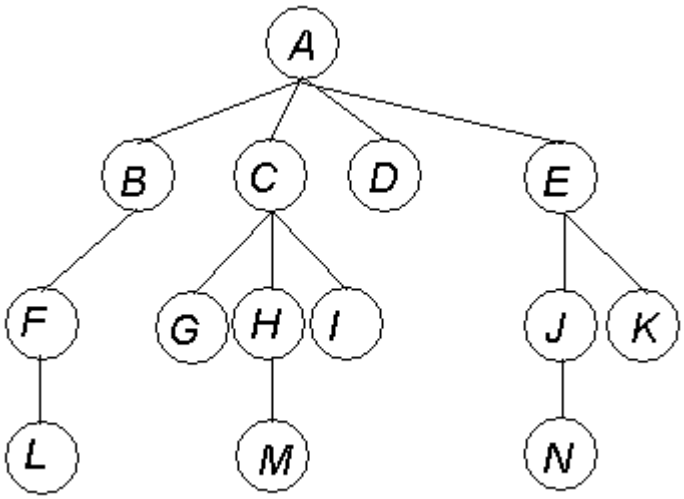
Επίσης οι γραμμικές λίστες μπορούν να αντιμετωπισθούν και άρα να υλοποιηθούν ως **στατικές** και ως **δυναμικές**. Στις στατικές γραμμικές λίστες

έχει προκαθοριστεί το ακριβές μέγεθος της κύριας μνήμης που θα χρησιμοποιείται για την αποθήκευση τους, ενώ στις δυναμικές ο χώρος που καταλαμβάνουν στην κύρια μνήμη είναι μεταβλητός και μεταβάλλεται ανάλογα με τις απαιτήσεις του προγράμματος.

35. Δέντρα

35.1 Εισαγωγή

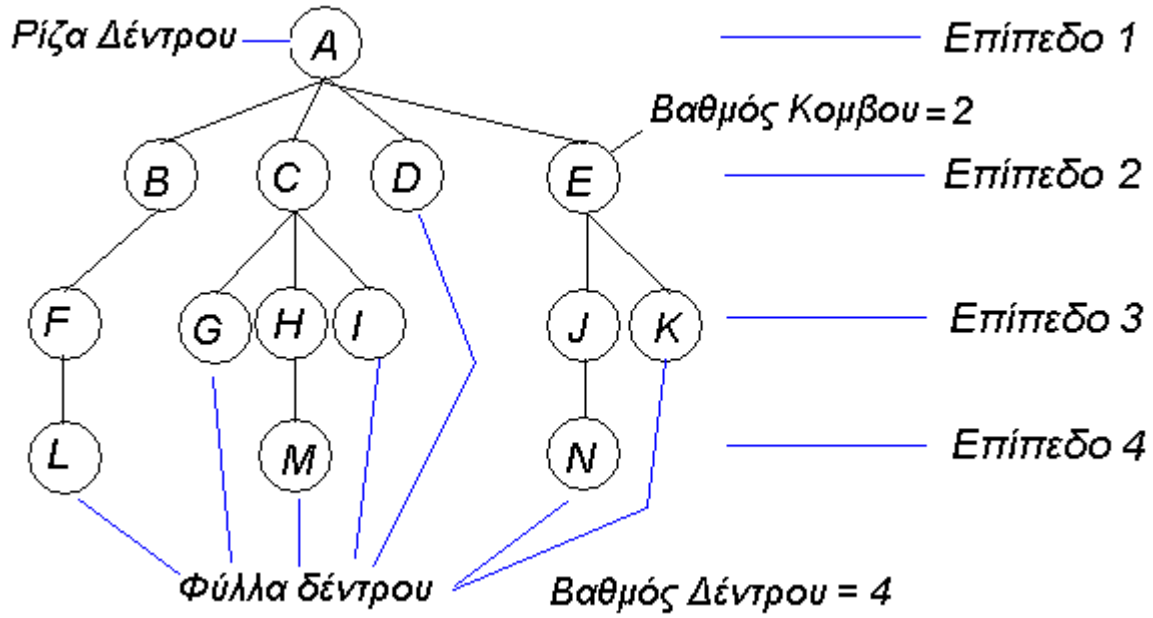
Τα δέντρα τα συναντούμε τις περισσότερες φορές όταν θέλουμε να αναπαραστήσουμε το γενεαλογικό μας δέντρο. Στον χώρο της πληροφορικής ο πιο συχνός τρόπος αναπαράστασης είναι αυτός που φαίνεται στο Σχήμα 1, όπου το δέντρο μεγαλώνει από τα πάνω προς τα κάτω.



Σχήμα 1: Παράδειγμα δέντρου

Είναι η δομή που εκφράζει κόμβους δεδομένων με ιεραρχική διάταξη. Στο πρώτο επίπεδο υπάρχει μόνο ένας κόμβος, **η ρίζα**. Κάθε κόμβος μπορεί να συνδέεται προς έναν ή περισσότερους κόμβους, τους **κόμβους παιδιά** του. Κάθε κόμβος έχει μόνο έναν **κόμβο πατέρα**, εκτός από τη ρίζα, η οποία είναι ο μόνος κόμβος χωρίς κόμβο πατέρα. Οι κόμβοι που δεν έχουν κόμβους παιδιά ονομάζονται φύλλα του δέντρου ή εξωτερικοί ή τερματικοί.

Το μεγαλύτερο επίπεδο στο οποίο μπορεί να βρίσκεται ένας κόμβος του δέντρου είναι το **βάθος** (ή ύψος) του δέντρου. **Βαθμός ενός κόμβου** του δέντρου είναι το πλήθος των κόμβων παιδιών του. **Βαθμός του δέντρου** είναι ο μέγιστος από τους βαθμούς όλων των κόμβων του δέντρου.

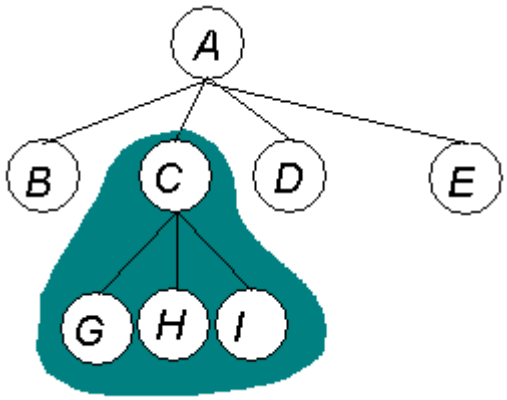


Σχήμα 2: Στοιχεία του δέντρου

Στο παραπάνω παράδειγμα το "A" είναι η ρίζα του δέντρου. Έχει τέσσερις κόμβους παιδιά, το "B", το "C" το "D", και το "E", οπότε ο βαθμός της

ρίζας είναι 4. Ο κόμβος "C" έχει τρεις κόμβους παιδιά (βαθμός κόμβου 3), το "G" , το "H" και το "I". Ο κόμβος "F" έχει ένα κόμβο παιδί, το "L". Οι κόμβοι "L", "G", "M", "I", "D", "N" και "K" είναι κόμβοι φύλλα (δεν έχουν κόμβους παιδιά). Ο βαθμός του δέντρου είναι 4 και το βάθος επίσης 4.

Κάθε κομμάτι του δέντρου από έναν κόμβο και κάτω ονομάζεται **υποδέντρο** του δέντρου. Στο ακόλουθο Σχήμα 3 ορίζεται ένα υποδέντρο στο δέντρο του παραπάνω παραδείγματος.



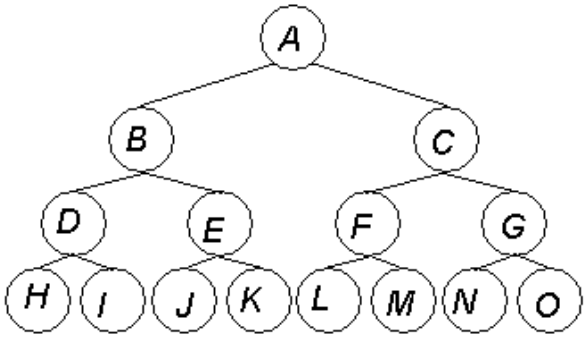
Σχήμα 3: Υποδέντρο του δέντρου

Όταν έχει σημασία η διάταξη των κλαδιών του δέντρου, τότε αυτό λέγεται **διατεταγμένο**. Η πιο ενδιαφέρουσα μορφή διατεταγμένου δέντρου είναι το δυαδικό δέντρο αναζήτησης.

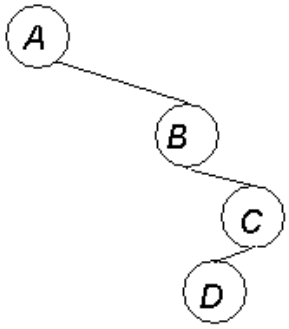
35.2 Δυαδικά Δέντρα

Τα δυαδικά δέντρα έχουν βαθμό 2, δηλαδή κάθε κόμβος μπορεί να έχει μέχρι δύο κόμβους παιδιά. Επομένως κάθε κόμβος μπορεί να έχει δύο υποδέντρα, το αριστερό υποδέντρο και το δεξί υποδέντρο.

Όταν κάθε εσωτερικός κόμβος του δέντρου έχει δύο και μόνο δύο υποδέντρα λέγεται **αυστηρά** δυαδικό δέντρο.



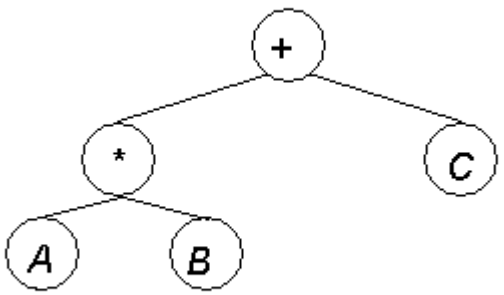
Σχήμα 1:Αυστηρό Δυαδικό Δέντρο.



Σχήμα 2:Ισχνό Δυαδικό Δέντρο.

Τα αυστηρά δυαδικά δέντρα με n φύλλα έχουν 2n-1 κόμβους. Αντίθετα **ισχνά** λέγονται τα δυαδικά δέντρα που κάθε εσωτερικός κόμβος έχει το πολύ ένα υποδέντρο. Το δέντρο Huffman είναι ένα αυστηρό δυαδικό δέντρο.

Το δυαδικό δέντρο χρησιμοποιείται πολλές φορές για την παράσταση αριθμητικών πράξεων Στο Σχήμα 1 που ακολουθεί, η απλή αλγεβρική έκφραση A*B+C μπορεί να παρασταθεί με ένα δυαδικό δέντρο που ονομάζεται δέντρο έκφρασης.



Σχήμα 3: Δέντρο Έκφρασης

35.3 Διάσχιση Δυαδικού Δέντρου

Όταν θέλουμε να κάνουμε διάσχιση (επίσκεψη όλων των κόμβων) του δέντρου, μπορούμε να ακολουθήσουμε τρεις τρόπους.

1ος Τρόπος

Ακολουθούμε τα εξής βήματα:

1. Επίσκεψη της Ρίζας
2. Επίσκεψη του αριστερού υποδέντρου
3. Επίσκεψη του δεξιού υποδέντρου

Αυτός ο τρόπος διάσχισης λέγεται **προδιατεταγμένος,** και με αυτή τη μέθοδο οι κόμβοι του δέντρου του Σχήματος 3 διασχίζονται με τη σειρά +*ABC. Αυτή η σειρά διάσχισης λέγεται **προθεματική** μορφή της έκφρασης A*B+C.

2ος Τρόπος

Ακολουθούμε τα εξής βήματα:

1. Επίσκεψη του αριστερού υποδέντρου
2. Επίσκεψη του δεξιού υποδέντρου
3. Επίσκεψη της Ρίζας

Αυτή ή διάσχιση λέγεται **μεταδιατεταγμένη,** και με αυτή τη μέθοδο οι κόμβοι του δέντρου του Σχήματος 3 διασχίζονται με τη σειρά AB*C+.

3ος Τρόπος

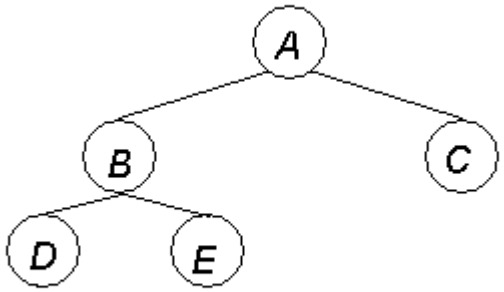
Ακολουθούμε τα εξής βήματα:

1. Επίσκεψη του αριστερού υποδέντρου
2. Επίσκεψη της Ρίζας
3. Επίσκεψη του δεξιού υποδέντρου

Αυτός ο τρόπος διάσχισης λέγεται **ενδοδιατεταγμένος,** και με αυτή τη μέθοδο οι κόμβοι του δέντρου του Σχήματος 3 διασχίζονται με τη σειρά A*B+C. Αυτή η σειρά διάσχισης ονομάζεται ένθετη μορφή της έκφρασης A*B+C.

35.4 Αποθήκευση Δυαδικών Δέντρων

Η αποθήκευση των δυαδικών δέντρων μπορεί να γίνει με τη χρήση πινάκων. Έστω ότι θέλουμε να αποθηκεύσουμε το δέντρο του Σχήματος 4.



Σχήμα 4: Δυαδικό δέντρο

Ο πίνακας που θα δημιουργηθεί για την αποθήκευση του είναι ο ακόλουθος.

1	2	3	4	5
A	B	C	D	E
2	4	0	0	0
3	5	0	0	0

Η πρώτη γραμμή του πίνακα είναι η αρίθμηση των κόμβων. Ο πίνακας διαβάζεται ως εξής: η πρώτη στήλη του πίνακα δηλώνει ότι ο κόμβος Α έχει παιδιά τους κόμβους που απορρυθμίζονται στον πίνακα ως 2 και 3, δηλαδή τους κόμβους Β και C κ.ο.κ.

Το μειονέκτημα αυτής της μεθόδου είναι ότι 6 κελιά του πίνακα είναι μηδενικά οπότε έχουμε μεγάλη σπατάλη μνήμης σε περίπτωση που το δέντρο δεν είναι **πλήρες**.

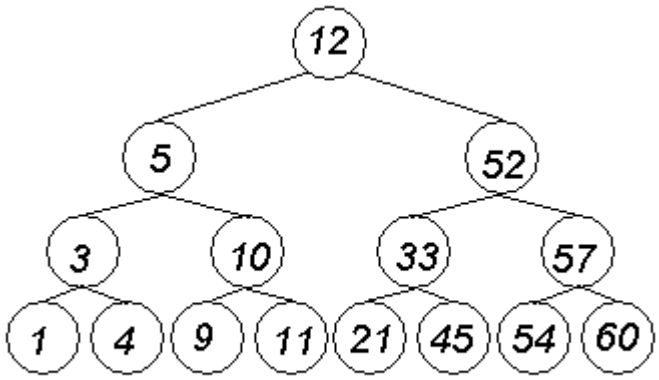
Ένας εναλλακτικός και ίσως καλύτερος τρόπος αποθήκευσης θα ήταν ο ακόλουθος πίνακας.

1	2	3	4	5
0	1	1	2	2

Ο πίνακας αυτός παρουσιάζει τον πατέρα του αντίστοιχου αριθμού του κόμβου. Ο κόμβος 1, που είναι ο Α, δεν έχει πατέρα κόμβο οπότε και παίρνει την τιμή 0. Αντίθετα ο κόμβος 2, που είναι το Β, έχει πατέρα κόμβο το Α γιαυτό και παίρνει το νούμερο 1.

35.5 Δυαδικά Δέντρα Αναζήτησης

Τα δυαδικά δέντρα αναζήτησης είναι μία ειδική κατηγορία διατεταγμένων δυαδικών δέντρων τα οποία είναι ταξινομημένα ως εξής: όλα τα αριστερά υποδέντρα κάθε κόμβου έχουν τιμές μικρότερες από τον ίδιο τον κόμβο ενώ όλα τα δεξιά μεγαλύτερες. Ένα παράδειγμα δυαδικού δέντρου αναζήτησης φαίνεται στο Σχήμα 5 που ακολουθεί.



Σχήμα 5 : Δυαδικό Δέντρο Αναζήτησης

Το δυαδικό δέντρο αναζήτησης μπορεί να χρησιμοποιηθεί ως μια μέθοδος τοποθέτησης και εύρεσης πληροφοριών σε μια βάση δεδομένων. Τα αρχεία αποθηκεύονται σε θέσεις που βρίσκονται στα φύλλα. Ο αλγόριθμος βρίσκει τις πληροφορίες διαιρώντας τον αριθμό των αρχείων που μένουν κάθε φορά μειώνοντας έτσι το πλήθος στο μισό έως ότου παραμένει μόνο ένα.

Ο διαδικασία που ακολουθεί ο αλγόριθμος του δυαδικού δέντρου αναζήτησης είναι ο ακόλουθος: Γίνεται έλεγχος της ρίζας. Αν η τιμή της ρίζας είναι ο αριθμός που ψάχνουμε τότε σταματάμε. Αν η τιμή τις ρίζας είναι μεγαλύτερη από την τιμή που ψάχνουμε τότε ακολουθούμε τη διαδρομή του αριστερού υποδέντρου, διαφορετικά ακολουθούμε τη διαδρομή του δεξιού.

Για παράδειγμα, έστω ότι ψάχνουμε τον κόμβο με τον αριθμό 45 στο Σχήμα 5. Ο αλγόριθμος ξεκινάει από τη ρίζα του δέντρου. Επειδή η τιμή της ρίζας είναι μικρότερη από την τιμή που ψάχνουμε, ακολουθούμε τη διαδρομή του δεξιού υποδέντρου. Κατόπιν προχωράμε στο αριστερό υποδέντρο διότι η

τιμή που ψάχνουμε είναι μικρότερη του 52 και έπειτα στο δεξί διότι το 45 είναι μεγαλύτερο του 33 απ'όπου και βρίσκουμε τον επιθυμητό κόμβο.

35.6 Δέντρα Huffman

Για να καταλάβουμε πώς η κωδικοποίηση Huffman μειώνει το μήκος ενός κωδικοποιημένου μηνύματος πρέπει πρώτα να δούμε πως κωδικοποιείται ένα μήνυμα.

35.6.1 Κωδικοποίηση Μηνύματος

Έστω ότι το αλφάβητο μας αποτελείται από τα τέσσερα σύμβολα A, B, C και D. Θέλουμε να κωδικοποιήσουμε το ακόλουθο μήνυμα «ABBABACDA». Χρησιμοποιώντας 3 bits για κάθε σύμβολο θα είχαμε την εξής κωδικοποίηση:

A = 010

B = 100

C = 000

D = 111

ABBABACDA = 010100100010100010000111010 δηλαδή 27 bits

Η αποκωδικοποίηση του μηνύματος θα ήταν εύκολη διότι κάθε γράμμα έχει σταθερό μήκος των 3 bits, οπότε κάθε 3 bits θα γνωρίζαμε ότι επρόκειτο για ένα γράμμα. Επειδή όμως έχουμε 4 σύμβολα η κωδικοποίηση θα ήταν καλύτερη από άποψης χώρου αν χρησιμοποιούσαμε 2 bits για κάθε σύμβολο. Οπότε θα είχαμε:

A = 00

B = 01

C = 10

D = 11

ABBABACDA = 000101000100101100 δηλαδή 18 bits.

35.6.2 Κωδικοποίηση Huffman

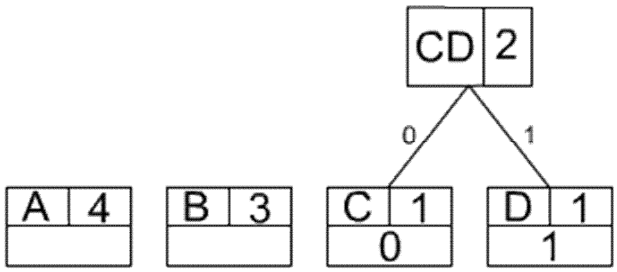
Με τη κωδικοποίηση Huffman επιδιώκουμε να ελαχιστοποιήσουμε το μέγεθος του μηνύματος που θέλουμε να κωδικοποιήσουμε. Για να το επιτύχουμε χρειάζεται να ληφθεί υπόψη πόσες φορές εμφανίζεται κάθε γράμμα στο μήνυμα . Στόχος είναι να αντικαταστήσουμε τα γράμματα που εμφανίζονται πιο συχνά με μικρότερους κωδικούς.

Στο μήνυμα «ABBABACDA» τα τέσσερα σύμβολα εμφανίζονται 4,3,1 και 1 φορές αντίστοιχα.

Σύμβολο	A	B	C	D
Συχνότητα	4	3	1	1

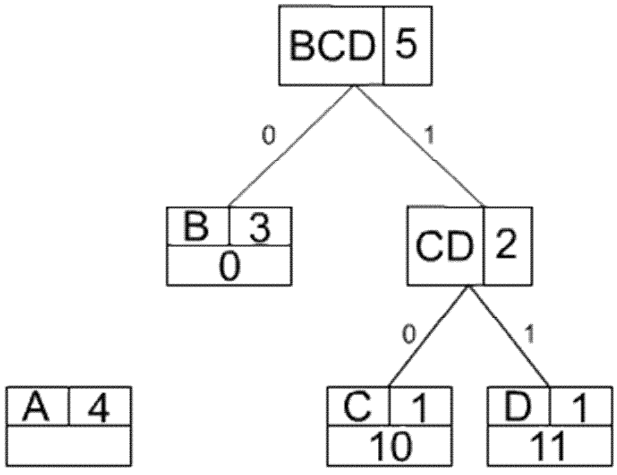
Η μέθοδος που ακολουθεί η κωδικοποίηση είναι η ακόλουθη:Πρώτα επιλέγονται τα πιο σπάνια σύμβολα, που είναι τα C και D. Αποφασίζεται ότι το τελευταίο ψηφίο τους θα είναι διαφορετικό, 0 για το C και 1 για το D. Τα δύο σύμβολα αυτά δίνουν το συνδυασμένο σύμβολο CD. Η συχνότητα του

συμβόλου CD είναι το άθροισμα των συχνοτήτων των συμβόλων C και D, δηλαδή 2. Η διαδικασία φαίνεται στο Σχήμα 1.



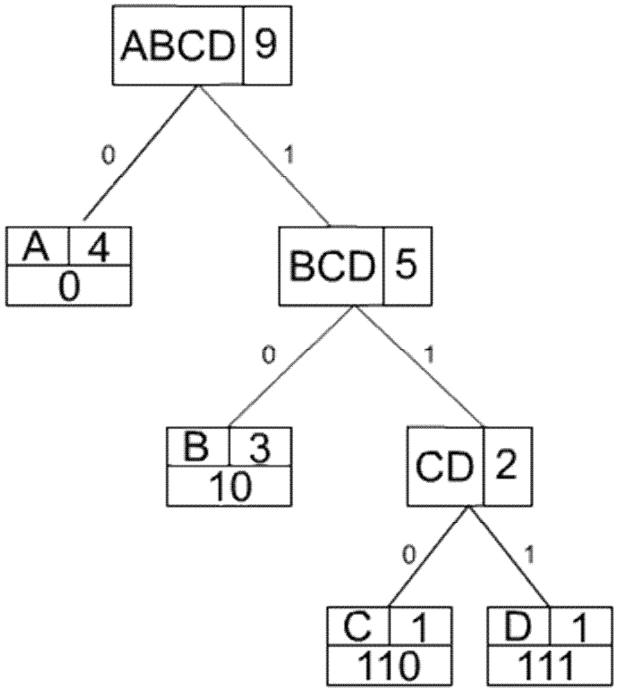
Σχήμα 1: Δημιουργία Δέντρου Huffman

Στη συνέχεια θεωρούμε ότι τα σύμβολα είναι 3, το A, B, και το CD.με συχνότητες 4, 3, και 2 αντίστοιχα. Επιλέγονται πάλι τα δύο πιο σπάνια σύμβολα που είναι το B και το CD. Τα δύο αυτά σύμβολα διαφοροποιούνται πάλι κατά το τελευταίο τους ψηφίο, 0 για το B και 1 για το CD.



Σχήμα 2: Δημιουργία Δέντρου Huffman

Ομοίως μένουν τα δύο σύμβολα A και BCD τα οποία διαφοροποιούνται κατά το τελευταίο τους ψηφίο.

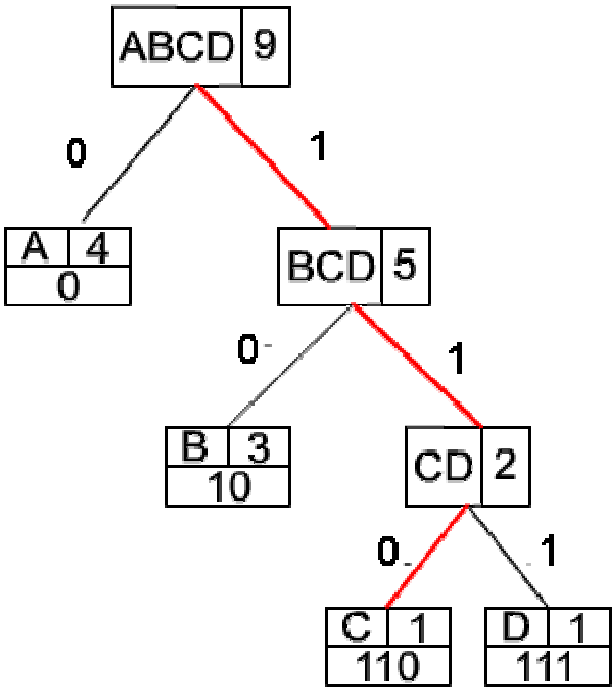


Σχήμα 3: Δημιουργία Δέντρου ***Huffman***

Η μέθοδος που χρησιμοποιεί η κωδικοποίηση Huffman συνδυάζοντας δύο σύμβολα κάθε φορά υποβάλλει τη χρήση αυστηρά δυαδικού δέντρου (κάθε κόμβος έχει δύο και μόνο δύο υποδέντρα) που φαίνεται στο Σχήμα 4.Το δέντρο αυτό ονομάζεται δέντρο Huffman.

Κάθε φύλλο παριστά ένα γράμμα του αρχικού αλφαβήτου, ενώ οι εσωτερικοί κόμβοι παριστούν ένα συνδυασμό γραμμάτων.

Για να βρούμε τον κωδικό κάποιου γράμματος αρκεί να ακολουθήσουμε τη διαδρομή από τη ρίζα έως ότου το βρούμε. Στο Σχήμα 4 βλέπουμε τη διαδρομή και τον κωδικό του C.



Σχήμα 4: Κωδικός του C.

Οι κωδικοί που προκύπτουν για κάθε σύμβολο είναι.

Σύμβολο	Κωδικός
A	0
B	10
C	110
D	111

Οπότε το μήνυμα «ABBABACDA» = 0101001001101110 δηλαδή 16 bits αντί 18 bits.

Μειώσαμε το μήνυμα κατά 11,25 % (Συμπίεση).

Οι κωδικοί που προκύπτουν από την κωδικοποίηση είναι μεταβλητού μήκους. Αυτό θα μπορούσε να παρουσιάσει πρόβλημα κατά την αποκωδικοποίηση. Εντούτοις με τη κωδικοποίηση Huffman οι κωδικοί που προκύπτουν είναι διαλεγμένοι με τέτοιον τρόπο που δεν αφήνουν περιθώρια σύγχυσης.

Το μήνυμα που θα αποκωδικοποιήσουμε θα είναι 0101001001101110

Όταν δούμε ότι ο κωδικός αρχίζει από 0 αυτόματα γνωρίζουμε ότι πρόκειται για το Α διότι κανένας άλλος κωδικός δεν αρχίζει από 0.

0		101001001101110
A		

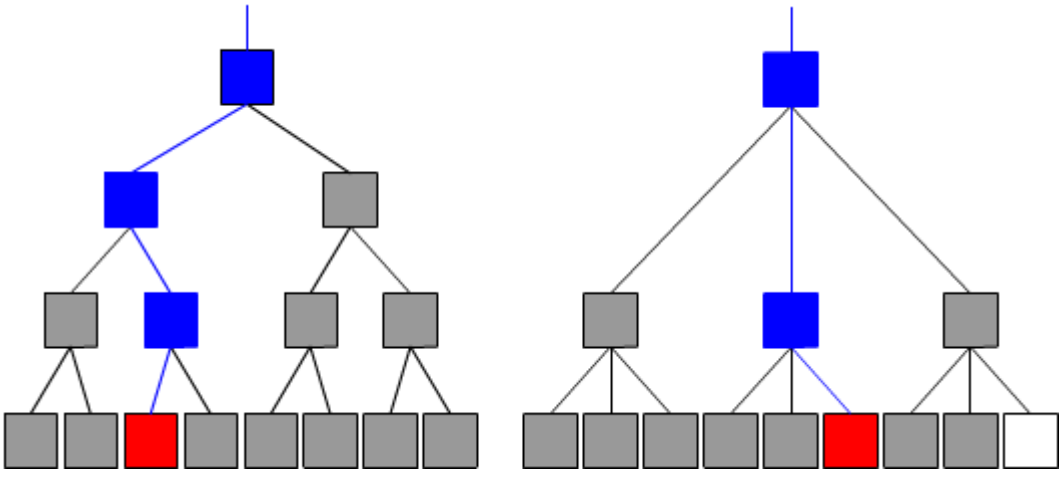
Στη συνέχεια τα bits που πρέπει να αποκωδικοποιήσουμε είναι τα 101001001101110. Επιδή αρχίζει από 1 ξέρουμε ότι το σύμβολο που θα προκύψει θα είναι ή Β ή C ή το.D. Επειδή όμως το επόμενο νούμερο είναι 0 γνωρίζουμε ότι μόνο το Β αρχίζει από 10. Άρα το επόμενο νούμερο είναι το Β.

010		1001001101110
AB		

Η διαδικασία συνεχίζεται εώς ότου αποκωδικοποιηθεί όλο το μήνυμα.

35.7 Β-Δέντρο

Η δομή του Β-δέντρου είναι τέτοια που επιλύει το πρόβλημα που παρουσιάζονται στα δέντρα αναζήτησης δηλαδή τη δυσκολία να παραμείνουν ισοζυγισμένα μετά από διάφορες εισαγωγές και διαγραφές κλειδιών. Ο αλγόριθμος του Β-δέντρου επιταχύνει τη διαδικασία εντοπισμού ενός επιθυμητού αρχείου με το να ελαχιστοποιεί τον αριθμό προσβάσεων σε μια βάση δεδομένων. Είναι γνωστό ότι παίρνει πολύ παραπάνω χρόνο η πρόσβαση μιας πληροφορίας στο σκληρό δίσκο παρά στη μνήμη, και αυτό οφείλεται στο γεγονός ότι ο δίσκος αποτελείται από μηχανικά μέρη, τα οποία διαβάζουν και γράφουν τα δεδομένα πολύ πιο αργά απ'ότι τα καθαρώς ηλεκτρονικά μέσα. Γι'αυτό και τα Β-δέντρα προτιμούνται όταν τα σημεία απόφασης, (οι κόμβοι) βρίσκονται στο σκληρό δίσκο παρά στη μνήμη. Τα Β-δέντρα γλιτώνουν χρόνο επειδή κάθε κόμβος έχει πολλούς κλάδους (δηλαδή παιδιά), σε σύγκριση με τα δυαδικά δέντρα, στα οποία κάθε κόμβος έχει μόνο δύο παιδιά. Όσο πιο πολλοί κόμβοι-παιδιά υπάρχουν τόσο λιγότερες προσβάσεις χρειάζονται να γίνουν. Ένα απλουστευμένο παράδειγμα αυτής της αρχής παρουσιάζεται στο Σχήμα 1:



Σχήμα 1:Σύγκριση προσβάσεων μεταξύ δυαδικών και Β-δέντρων.

Σε ένα δέντρο, τα αρχεία αποθηκεύονται στις θέσεις που βρίσκονται τα φύλλα του δέντρου. Ο αριθμός προσβάσεων που απαιτούνται για να φθάσουν στο επιθυμητό αρχείο είναι ίσος με το βάθος του δέντρου. Η εικόνα στα αριστερά παρουσιάζει ένα δυαδικό δέντρο που χρησιμοποιείται για την εντόπιση ενός συγκεκριμένου αρχείου σε ένα σύνολο οκτώ φύλλων. Η εικόνα στα δεξιά παρουσιάζει ένα B-δέντρο για την εντόπιση ενός αρχείου σε ένα σύνολο οκτώ φύλλων (το ένα φύλλο δεν περιέχει αρχείο και ονομάζεται null). Το δυαδικό δέντρο στα αριστερά έχει βάθος τέσσερα ενώ το B-δέντρο στο δεξιά έχει βάθος τρία. Σαφώς, το B-δέντρο μπορεί να βρει ένα επιθυμητό αρχείο κάνοντας λιγότερες προσβάσεις. Το μειονέκτημα είναι ότι η διαδικασία απόφασης σε κάθε κόμβο είναι πιο περίπλοκη σε ένα B-δέντρο παρά σε ένα δυαδικό δέντρο.

Τα B-δέντρα βαθμού d έχουν τα εξής χαρακτηριστικά:

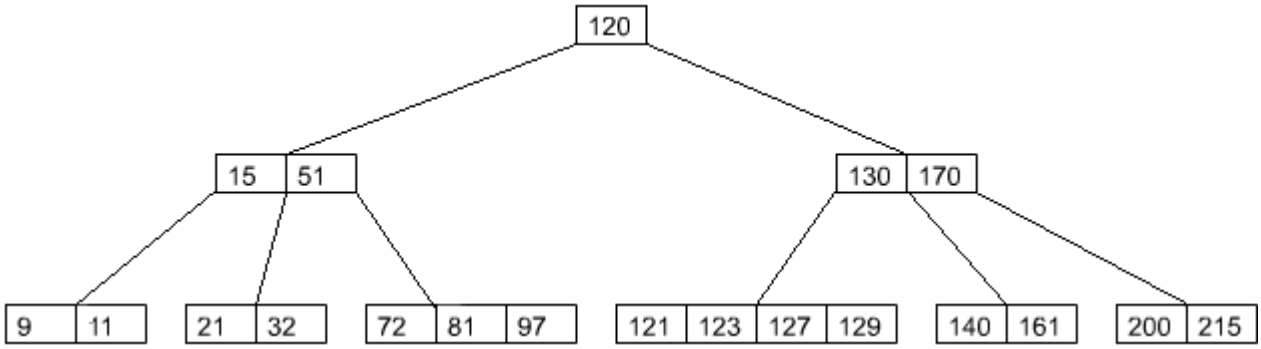
1. η ρίζα έχει το ελάχιστο ένα κλειδί και μέγιστο 2d κλειδιά

2. κάθε εσωτερικός κόμβος έχει ελάχιστο d κλειδιά και μέγιστο 2d κλειδια

3. ένας κόμβος με k-κλειδιά έχει k+1 παιδιά.

4. όλα τα φύλλα βρίσκονται στο ίδιο επίπεδο

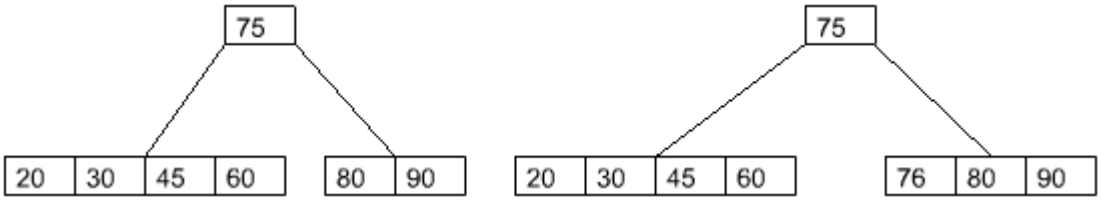
Στο Σχήμα 2 παρουσιάζεται ένα B-δέντρο βαθμού 2 με 3 επίπεδα. Παρόλο που οι κόμβοι έχουν 2,3 ή 4 παιδιά η ρίζα αποτελεί εξαίρεση και περιέχει 1 κλειδί.



Σχήμα 2:Β-δέντρο βαθμού 2 με 3 επίπεδα.

35.7.1 Εισαγωγή κλειδιών στα Β-δέντρα

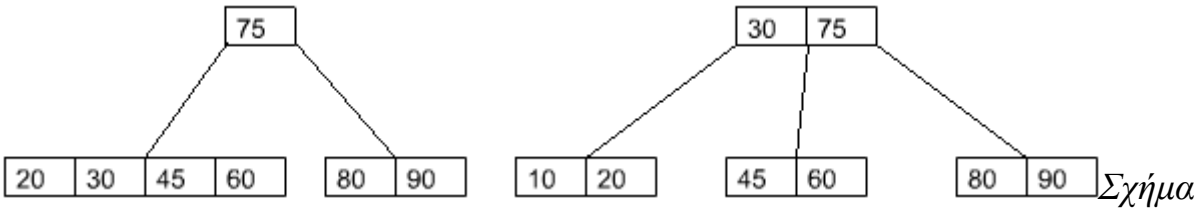
Η εισαγωγή κλειδιών στα Β-δέντρα γίνεται πάντα στα φύλλα του. Η εισαγωγή κλειδιού στα Β-δέντρα μπορεί να απαιτεί να γίνει ανακατανομή του δέντρου. Δυσκολία στην εισαγωγή κλειδιού παρουσιάζεται στη περίπτωση που στον κόμβο που θέλουμε να εισάγουμε βρίσκονται 2d κλειδιά (που είναι ο μέγιστος αριθμών κλειδιών ανά κόμβο) Για να καταλάβουμε τη διαφορά παρακάτω αναλύονται οι δύο περιπτώσεις εισαγωγής κόμβων στα Β-δέντρα. Στο Σχήμα 3 που ακολουθεί παρουσιάζεται το αποτέλεσμα της εισαγωγής του κλειδιού 76 στο Β-δέντρο του οποίου ο κόμβος στον οποίο θα εισαχθεί το κλειδί έχει λιγότερο από 2d κλειδιά. (Εδώ ο βαθμός του δέντρου είναι d = 2).



Σχήμα 3:Εισαγωγή του κλειδιού 76 σε κόμβο με λιγότερα από 2d κλειδιά

Αντίθετα στην περίπτωση που θέλουμε να εισάγουμε το κλειδί σε ένα κόμβο που περιέχει ήδη 2d κλειδιά, η εισαγωγή είναι αδύνατη (παρουσιάζεται υπερχείλιση). Οπότε ο κόμβος διασπάται σε δύο με τη δημιουργία ενός νέου

κόμβου. Τα 2d+1 κλειδιά μοιράζονται από d κόμβους ενώ το μεσαίο κλειδί ανεβαίνει στο κόμβο πατέρα. Στο Σχήμα 4 που ακολουθεί παρουσιάζεται η εισαγωγή του κλειδιού 10 σε έναν κόμβο που περιέχει 2d κλειδιά.

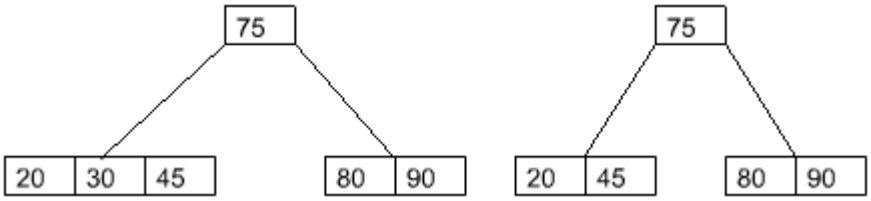


Σχήμα 4: Εισαγωγή του κλειδιού 10 σε κόμβο με 2d κλειδιά

Στην πιο ακραία περίπτωση μπορεί να έχουμε διάσπαση της ρίζας, οπότε και το ύψος του δέντρου αυξάνει κατά ένα.

35.7.2 Διαγραφή κλειδιού στα B-δέντρα

Η διαδικασία διαγραφής στα B-δέντρα εξαρτάται από την περίπτωση που ο κόμβος που περιέχει το κλειδί είναι φύλλο ή όχι. Αν ο κόμβος που πρόκειται να διαγραφεί ανήκει σε εσωτερικό κόμβο τότε τον αντικαταστούμε με το αμέσως επόμενο εσωτερικό κόμβο. Επιπλέον η διαγραφή του κλειδιού εξαρτάται από το αν το φύλλο που ανήκει το κλειδί περιέχει συνολικά περισσότερα από d-κλειδιά. Στην περίπτωση αυτή η διαγραφή του κλειδιού είναι εύκολη. Στο Σχήμα 5 που ακολουθεί φαίνεται η διαγραφή του κλειδιού 30 από κόμβο που περιέχει περισσότερα από d-κλειδιά.

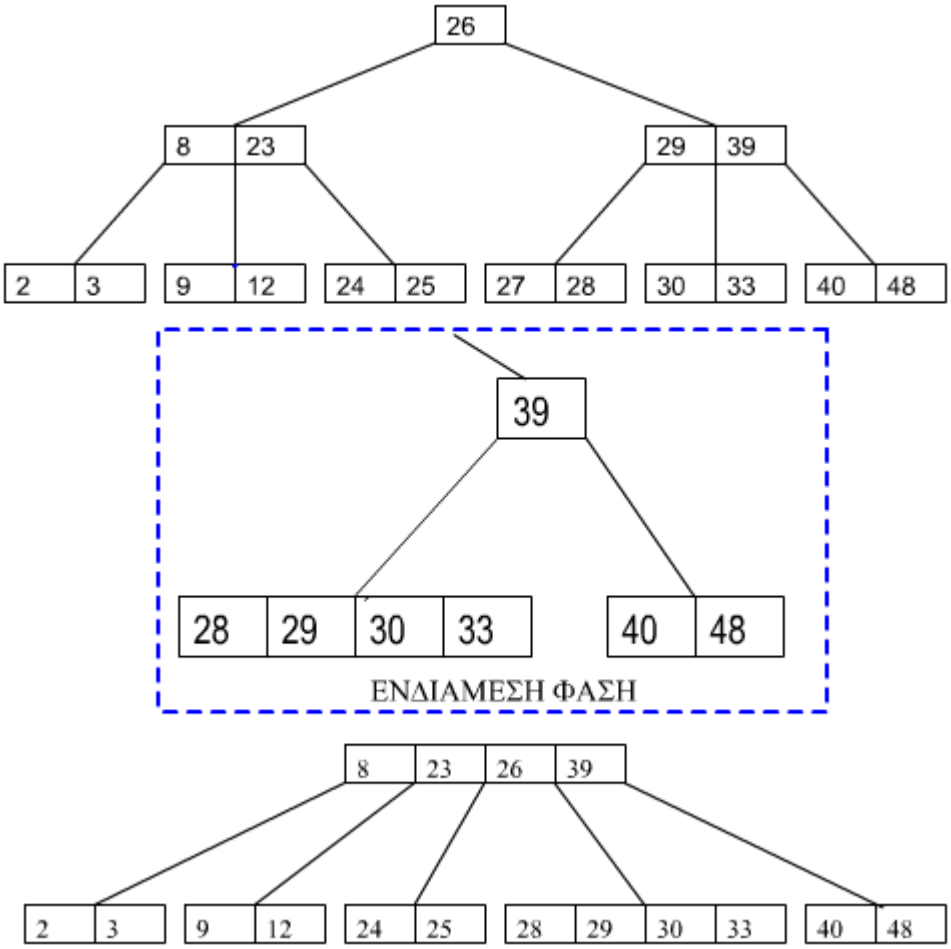


Σχήμα 5: Διαγραφή του κλειδιού 30 από κόμβο που περιέχει περισσότερα από d-κλειδιά

Αντίθετα στη περίπτωση που το φύλλο περιέχει ακριβώς d-κλειδιά, τότε με τη διαγραφή κλειδιού δημιουργείται πρόβλημα λόγω υποχείλησης και αντιμετωπίζεται ανάλογα με έναν από δύο τους ακόλουθους τρόπους.

Η πρώτη περίπτωση εφαρμόζεται όταν ο δεξιός γειτονικός κόμβος περιέχει περισσότερα από d-κλειδιά. Τότε η διαγραφή αντιμετωπίζεται με ισοζυγισμό των κόμβων.

Η δεύτερη περίπτωση εφαρμόζεται όταν ο δεξιός γειτονικός κόμβος περιέχει ακριβώς d-κλειδιά. Τότε η διαγραφή αντιμετωπίζεται με συγχώνευση των κόμβων. Το Σχήμα 6 που ακολουθεί παρουσιάζεται η διαγραφή του κλειδιού 27 από κόμβο που περιέχει d-κλειδιά και ο δεξιός γειτονικός κόμβος περιέχει ακριβώς d-κλειδιά.



Σχήμα 6:διαγραφή του κλειδιού 27

Οποιαδήποτε από τις δύο περιπτώσεις εφαρμόζουμε όταν το κλειδί που πρόκειται να διαγραφεί ανήκει σε κόμβο που δεν έχει δεξιό γείτονα τότε ελέγχεται ο αριστερός. Επίσης στη διαδικασία συμμετέχουν και τα κλειδιά του κόμβου πατέρα ώστε να μην διαταραχθεί η λεξικογραμμική σειρά των κλειδιών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Θραμπουλίδης, Κλεάνθης Χρ., Από τη C στη Java : από τον διαδικαστικό στον object-oriented προγραμματισμό, Εκδόσεις Τζιόλας, Θεσσαλονίκη, 1999.

2. Lafore Robert, Αλγόριθμοι και δομές δεδομένων στη Java, Εκδόσεις Γκιούρδας Μ., Αθήνα 2005.

3. Μανωλόπουλος Γ., Δομές Δεδομένων, Τόμος Α, Εκδόσεις Art of Text, Θεσ/νίκη, 1991.

4. Βλέτσας γ., Διαχείριση Δεδομένων, Αθήνα 1987

5. Wirth N., Αλγόριθμοι & Δομές Δεδομένων, Εκδόσεις Κλειδάριθμος, Αθήνα, 1990

ΗΛΕΚΤΡΟΝΙΚΕΣ ΠΗΓΕΣ

www.aueb.gr

Οικονομικό Πανεπιστήμιο Αθηνών

ate.teiath.gr

ΤΕΙ Αθήνας

www.ntua.gr

Εθνικό Μετσόβιο Πολυτεχνείο

www.uoa.gr

Εθνικό και Καποδιστριακό Πανεπιστήμιον Αθηνών

www.wikipedia.org