Δομές Δεδομένων - Εργασία 1

Δανοπούλου Έμυ (3170033)

Μπαλή Νίκη (3170114)

Μέρος Α

StringStackImpl

Η υλοποίηση της διεπαφής StringStack έγινε με την κλάση StringStackImpl, χρησιμοποιώντας λίστα μονής σύνδεσης. Ξεκινώντας, δηλώνουμε μια ακέραια μεταβλητή elements η οποία θα χρησιμοποιηθεί ως μετρητής των στοιχείων που υπάρχουν στη στοίβα, μετά από κάθε εισαγωγή ή εξαγωγή στοιχείου.

Στη συνέχεια δηλώνουμε μια μεταβλητή head τύπου Node η οποία αποτελεί τον δείκτη κεφαλής και δείχνει πάντα στο πρώτο στοιχείο της στοίβας, δηλαδή σε αυτό που προστέθηκε τελευταίο.

Έπειτα έχουμε μια κλάση Node η οποία κατασκευάζει τους κόμβους οι οποίοι θα αποτελέσουν την στοίβα.

Η μεταβλητή item τύπου String αποτελεί τα δεδομένα του στοιχείου και η μεταβλητή next τύπου Node αποτελεί τον σύνδεσμο με τον αμέσως επόμενο κόμβο στη στοίβα. Για να δημιουργήσουμε έναν καινούριο κόμβο, αρκεί να θέσουμε ως ορίσματα τα δεδομένα του και τον επόμενο κόμβο με τον οποίο θα συνδεθεί.

Ο constructor της κλάσης StringStackImpl παίρνει ως όρισμα έναν ακέραιο αριθμό maxN (ο οποίος δεν χρησιμοποιείται πουθενά αλλού) και θέτουμε *head == null* έτσι ώστε να δημιουργηθεί μια κενή στοίβα.

Η μέθοδος isEmpty() επιστρέφει την αντίστοιχη boolean τιμή για την συνθήκη head == null. Εάν η στοίβα είναι κενή, θα επιστραφεί true και εάν η στοίβα δεν είναι κενή, θα επιστραφεί false.

Η μέθοδος push παίρνει ως όρισμα μία τιμή String και δεν επιστρέφει τίποτα, αλλά εισάγει ένα αντικείμενο Node στην αρχή της λίστας. Το όρισμα item αποτελεί τα δεδομένα που περιέχει ο νέος κόμβος. Με κάθε κλήση της μεθόδου push, η μεταβλητή elements αυξάνεται κατά ένα, διότι προστίθεται ένα καινούριο στοιχείο στη στοίβα.

Η μέθοδος pop απωθεί ένα στοιχείο από τη στοίβα, αφαιρώντας τον κόμβο από την αρχή της λίστας (*head = head.next;*) και επιστρέφοντας το στοιχείο του (*String v = head.item; return v;*). Με κάθε κλήση της μεθόδου pop, η μεταβλητή elements μειώνεται κατά ένα, διότι αφαιρείται ένα στοιχείο από τη στοίβα. Εάν η στοίβα είναι άδεια, προκαλείται μία εξαίρεση τύπου NoSuchElementException και τυπώνεται το μήνυμα «The list is empty.».

Η μέθοδος peek επιστρέφει το στοιχείο που βρίσκεται στο εμπρός τμήμα της λίστας (δηλαδή στην κορυφή της στοίβας), χωρίς όμως να το αφαιρεί. Εάν η στοίβα είναι άδεια, προκαλείται μία εξαίρεση τύπου NoSuchElementException και τυπώνεται το μήνυμα «The list is empty.».

Η μέθοδος printStack παίρνει ένα όρισμα stream τύπου PrintStream και μέσω ενός while loop εκτυπώνει όλα τα στοιχεία της στοίβας.

Η μέθοδος size επιστρέφει την ακέραιη μεταβλητή elements η οποία αντιπροσωπεύει το πλήθος των στοιχείων της στοίβας.

IntQueueImpl

Η υλοποίηση της διεπαφής IntQueue έγινε με την κλάση IntQueueImpl, χρησιμοποιώντας λίστα μονής σύνδεσης. Ξεκινώντας, δηλώνουμε μια ακέραια μεταβλητή elements η οποία θα χρησιμοποιηθεί ως μετρητής των στοιχείων που υπάρχουν στην ουρά, μετά από κάθε εισαγωγή ή εξαγωγή στοιχείου.

Στη συνέχεια δηλώνουμε δύο μεταβλητές head και tail τύπου Node οι οποίες δείχνουν πάντα στο πρώτο και στο τελευταίο στοιχείο της ουράς, αντίστοιχα. Στις ουρές FIFO, το πρώτο στοιχείο είναι αυτό που εισήχθη πρώτο (δηλαδή το παλαιότερο) και το τελευταίο στοιχείο είναι αυτό που εισήχθη τελευταίο (δηλαδή το νεότερο).

Έπειτα έχουμε μια κλάση Node η οποία κατασκευάζει τους κόμβους οι οποίοι θα αποτελέσουν την ουρά.

Η μεταβλητή item τύπου String αποτελεί τα δεδομένα του στοιχείου και η μεταβλητή next τύπου Node αποτελεί τον σύνδεσμο με τον αμέσως επόμενο κόμβο στην ουρά. Για να δημιουργήσουμε έναν καινούριο κόμβο, αρκεί να θέσουμε ως ορίσματα τα δεδομένα του και τον επόμενο κόμβο με τον οποίο θα συνδεθεί.

Ο constructor της κλάσης IntQueueImpl παίρνει ως όρισμα έναν ακέραιο αριθμό maxN (ο οποίος δεν χρησιμοποιείται πουθενά αλλού) και θέτουμε *head == null* και *tail == null* έτσι ώστε να δημιουργηθεί μια κενή ουρά.

Η μέθοδος isEmpty() επιστρέφει την αντίστοιχη boolean τιμή για την συνθήκη head == null. Εάν η στοίβα είναι κενή, θα επιστραφεί true και εάν η στοίβα δεν είναι κενή, θα επιστραφεί false.

Η μέθοδος put εισάγει ένα καινούριο στοιχείο στην ουρά ορίζοντας τον δείκτη tail ώστε να δείχνει στο νέο στοιχείο (*tail = new Node(item);*) και εάν η ουρά δεν είναι άδεια, ενημερώνουμε τον δείκτη tail. Εάν όμως η ουρά είναι άδεια, τότε οι δείκτες head και tail ταυτίζονται. Με κάθε κλήση της μεθόδου put, η μεταβλητή elements αυξάνεται κατά ένα, διότι προστίθεται ένα καινούριο στοιχείο στην ουρά.

Η μέθοδος get αφαιρεί ένα στοιχείο από την ουρά, αφαιρώντας το στοιχείο που βρίσκεται στην αρχή της λίστας (*head = head.next;*) και επιστρέφοντας το στοιχείο του (*int v = head.item; return v;*). Με κάθε κλήση της μεθόδου get, η μεταβλητή elements μειώνεται κατά ένα, διότι αφαιρείται ένα στοιχείο από την ουρά. Εάν η ουρά είναι άδεια, προκαλείται μία εξαίρεση τύπου NoSuchElementException και τυπώνεται το μήνυμα «The queue is empty.».

Η μέθοδος peek επιστρέφει το παλαιότερο στοιχείο της ουράς, χωρίς όμως να το αφαιρεί. Εάν η ουρά είναι άδεια, προκαλείται μία εξαίρεση τύπου NoSuchElementException και τυπώνεται το μήνυμα «The queue is empty.».

Η μέθοδος printStack εκτυπώνει όλα τα στοιχεία της ουράς.

Η μέθοδος size επιστρέφει την ακέραιη μεταβλητή elements η οποία αντιπροσωπεύει το πλήθος των στοιχείων της ουράς.

Μέρος Β

Η κλάση TagMatching κληρονομεί την τάξη StringStackImpl και κατά συνέπεια όλες τις μεθόδους της.

private static StringStackImpl s1 = new StringStackImpl(0); : δημιουργούμε μία καινούρια άδεια στοίβα s1 με την οποία θα δουλέψουμε.

private static File f = null; private static BufferedReader reader = null; : δημιουργούμε δύο κενά αντικείμενα File και BufferedReader τα οποία θα χρησιμοποιήσουμε για να βρούμε, να ανοίξουμε και να διαβάσουμε το αρχείο.

TagMatching(int maxN){super(maxN); : constructor της κλάσης TagMatching ο οποίος κληρονομεί την παράμετρο maxN από την τάξη StringStackImpl.

public static boolean hasMatchingTags(File file) {…} : η μέθοδος αυτή επιστρέφει true εάν το αρχείο έχει ταιριασμένες ετικέτες και false εάν το αρχείο δεν έχει ταιριασμένες ετικέτες.

Η υλοποίηση της hasMatchingTags ξεκινά διαβάζοντας την πρώτη γραμμή του αρχείου και μετά μπαίνει σε ένα while loop για όσο το αρχείο έχει ακόμη γραμμές για να διαβαστούν. Βγαίνουμε από το loop όταν φτάσουμε στο τέλος του αρχείου. Έπειτα ορίζουμε έναν StringTokenizer με όρισμα την γραμμή που διαβάστηκε τελευταία και θέτουμε στην μεταβλητή token το πρώτο token αυτής της γραμμής. Όσο έχουμε tokens να διαβάσουμε από αυτή τη γραμμή, εξετάζουμε τα tokens τα οποία ξεκινούν με τον χαρακτήρα <.

s1.push(token.substring(1)); : Εάν το token ξεκινά μόνο με < τότε παίρνουμε ένα substring ξεκινώντας από τον δεύτερο χαρακτήρα του token και το κάνουμε push στη στοίβα s1.

if (token.substring(2).equals(s1.peek())){ s1.pop(); } : Εάν το token ξεκινά με </ τότε παίρνουμε ένα substring ξεκινώντας από τον τρίτο χαρακτήρα του token, το συγκρίνουμε με το πιο πρόσφατο στοιχείο της στοίβας s1 και εάν είναι τα ίδια, τότε αφαιρούμε το πιο πρόσφατο στοιχείο της στοίβας s1 μέσω της μεθόδου pop.

if (str.hasMoreTokens()){ token = str.nextToken();} else{ token = null; } : εάν ο StringTokenizer έχει ακόμη tokens για να διαβάσει, τότε προχωράμε στο επόμενο token και αν όχι, τότε θέτουμε *token = null;* και βγαίνουμε από το while loop.

line = reader.readLine(); : διαβάζει την επόμενη γραμμή κειμένου, εάν υπάρχει.

if (s1.isEmpty()) { return true; } : εάν στο τέλος της μεθόδου η στοίβα s1 είναι άδεια, σημαίνει πως όλα τα opening tags βρήκαν το closing tag που τους αντιστοιχεί και άρα επιστρέφεται τιμή true.

else{ return false; } : εάν στο τέλος της μεθόδου η στοίβα s1 δεν είναι άδεια, σημαίνει πως ένα τουλάχιστον opening tag δεν βρήκε το closing tag που του αντιστοιχεί και άρα επιστρέφεται τιμή false.

Εάν υπάρξει οποιοδήποτε πρόβλημα ανάγνωσης του αρχείου, δημιουργείται μια εξαίρεση και τυπώνεται το μήνυμα «Error reading file.» και επιστρέφεται τιμή false.

Μέθοδος main:

try { f = new File(args[0]); }

catch (NullPointerException e) { System.err.println ("File not found."); }

Δοκιμάζουμε να βρούμε το αρχείο που έχουμε λάβει ως όρισμα από τη γραμμή εντολών. Εάν δεν βρεθεί, προκαλείται μια εξαίρεση NullPointerException και τυπώνεται το μήνυμα «File not found.».

try { reader = new BufferedReader(new FileReader(f)); }

catch (FileNotFoundException e) { System.err.println("Error opening file. Terminating."); System.exit(1); }

Δοκιμάζουμε να ανοίξουμε το αρχείο. Εάν προκύψει κάποιο σφάλμα, προκαλείται μια εξαίρεση FileNotFoundException, τυπώνεται το μήνυμα «Error opening file. Terminating.» και το πρόγραμμα τερματίζεται.

if (hasMatchingTags(f)){ System.out.println("The file has matching tags."); }

else { System.out.println("The file doesn't have matching tags."); }

Εάν το αρχείο έχει ταιριασμένες ετικέτες, εκτυπώνεται το μήνυμα «*The file has matching tags.»*

Εάν το αρχείο δεν έχει ταιριασμένες ετικέτες, εκτυπώνεται το μήνυμα «*The file doesn't have matching tags.»*

try { reader.close(); }

catch (IOException e){ System.err.println("Error closing file."); }

Δοκιμάζουμε να κλείσουμε το αρχείο. Εάν προκύψει κάποιο σφάλμα, προκαλείται μια εξαίρεση IOException και τυπώνεται το μήνυμα «Error closing file.».

Μέρος Γ

Η κλάση NetBenefit κληρονομεί την τάξη IntQueueImpl και κατά συνέπεια όλες τις μεθόδους της.

private static IntQueueImpl queue = new IntQueueImpl(0); : δημιουργούμε μία καινούρια άδεια ουρά queue με την οποία θα δουλέψουμε.

private static File f = null; private static BufferedReader reader = null; : δημιουργούμε δύο κενά αντικείμενα File και BufferedReader τα οποία θα χρησιμοποιήσουμε για να βρούμε, να ανοίξουμε και να διαβάσουμε το αρχείο.

NetBenefit(int maxN){super(maxN); : constructor της κλάσης NetBenefit ο οποίος κληρονομεί την παράμετρο maxN από την τάξη IntQueueImpl.

Μέθοδος main:

try { f = new File(args[0]); }

catch (NullPointerException e) { System.err.println ("File not found."); }

Δοκιμάζουμε να βρούμε το αρχείο που έχουμε λάβει ως όρισμα από τη γραμμή εντολών. Εάν δεν βρεθεί, προκαλείται μια εξαίρεση NullPointerException και τυπώνεται το μήνυμα «File not found.».

try { reader = new BufferedReader(new FileReader(f)); }

catch (FileNotFoundException e) { System.err.println("Error opening file!"); }

Δοκιμάζουμε να ανοίξουμε το αρχείο. Εάν προκύψει κάποιο σφάλμα, προκαλείται μια εξαίρεση FileNotFoundException και τυπώνεται το μήνυμα «Error opening file.».

Διαβάζουμε την πρώτη γραμμή του αρχείου και μετά δημιουργούμε ένα αντικείμενο Scanner με όρισμα τη γραμμή αυτή.

int profit = 0; : αρχικοποιούμε μία ακέραια μεταβλητή profit η τιμή της οποίας θα αποτελεί το κέρδος ή τη ζημία της επιχείρησης από την πώληση μετοχών.

Έπειτα μπαίνουμε σε ένα while loop για όσο το αρχείο έχει ακόμη γραμμές για να διαβαστούν.

Εάν το πρώτο token που διαβάζει ο Scanner ισούται με τη λέξη buy, τότε ορίζουμε μια ακέραιη μεταβλητή n που θα έχει τον επόμενο ακέραιο αριθμό μετά τη λέξη buy. Αυτός ο αριθμός προφανώς αποτελεί την ποσότητα μετοχών που αγόρασε η εταιρεία. Στη συνέχεια, ορίζουμε μία ακόμα ακέραιη μεταβλητή price που θα έχει τον επόμενο ακέραιο αριθμό μετά τον αριθμό n. Αυτός ο αριθμός προφανώς αποτελεί την τιμή των μετοχών που αγόρασε η εταιρεία.

Μετά, μέσω ενός while loop, εισάγουμε n καινούρια στοιχεία στην ουρά queue, το καθένα με περιεχόμενο την τιμή price.

Εάν το πρώτο token που διαβάζει ο Scanner δεν ισούται με τη λέξη buy, τότε ορίζουμε μια ακέραιη μεταβλητή n που θα έχει τον επόμενο ακέραιο αριθμό που συναντάμε. Αυτός ο αριθμός προφανώς αποτελεί την ποσότητα μετοχών που θέλει να πουλήσει η εταιρεία.

Εάν αυτός το αριθμός n είναι μεγαλύτερος από το τρέχον μέγεθος της ουράς, τότε εκτυπώνονται τα μηνύματα:

"Number of stocks available for sale: "+queue.size()

"Number of stocks that are wanted for sale: "+n

"Error. Number of stocks for sale is out of bounds with the current number of stocks available"

Εάν όμως αυτός το αριθμός n δεν είναι μεγαλύτερος από το τρέχον μέγεθος της ουράς, τότε ορίζουμε μία ακόμα ακέραιη μεταβλητή price που θα έχει τον επόμενο ακέραιο αριθμό που συναντάμε. Αυτός ο αριθμός προφανώς αποτελεί την τιμή για την οποία σκοπεύει να πουλήσει η εταιρεία κάποιες από τις μετοχές της.

Μετά, μέσω ενός while loop, υπολογίζεται το κέρδος ή τη ζημία της εταιρείας από την πιο πρόσφατη πώληση μετοχών.

System.out.println("The"+((profit>=0) ? " profit" : " damage") +" is: "+profit); : τυπώνεται το κατάλληλο μήνυμα για το κέρδος ή τη ζημία της εταιρείας.

line = reader.readLine(); : διαβάζει την επόμενη γραμμή.

if (line != null){ in = new Scanner(line); } : εάν υπάρχει ακόμα κάποια γραμμή για να διαβαστεί, την θέτει ως όρισμα του Scanner.

in.close(); : κλείνει ο Scanner.

catch (IOException e){ System.err.println("Error reading file."); } : Εάν υπάρξει οποιοδήποτε πρόβλημα ανάγνωσης του αρχείου, δημιουργείται μια εξαίρεση και τυπώνεται το μήνυμα «Error reading file.».

try { reader.close(); reader = null; }

catch (IOException e){ System.err.println("Error closing file."); }

Δοκιμάζουμε να κλείσουμε το αρχείο. Εάν προκύψει κάποιο σφάλμα, προκαλείται μια εξαίρεση IOException και τυπώνεται το μήνυμα «Error closing file.».