ΑΝΑΦΟΡΑ ΠΑΡΑΔΩΣΗΣ:Εργασία 1, Δομές Δεδομένων

Μέρος α)

public interface IntQueue <Gene> {…}

public interface StringStack <Gene> {…}

Χρησιμοποιήσαμε Generics(όπου <Gene>) για να καταστήσουμε την ουρά και την στοίβα κατάλληλη για κάθε τύπο αντικειμένων.

public class StringStackImpl <Gene> implements StringStack <Gene> {…}

public class IntQueueImpl <Gene> implements IntQueue <Gene> {…}

Σαν ιδιότητες ορίσαμε τον πρώτο(First) και τον τελευταίο κόμβο(Last) της ουράς, και τον κορυφαίο κόμβο(Top) για την στοίβα και έναν κατασκευαστή χωρίς ορίσματα, το μέγεθος(size), την μέθοδο isEmpty(), που ελεγχει αν η δομή είναι άδεια βάση του τρέχοντος μεγέθους της και μια inner private class Node(με χαρακτηριστικά ένα αντικείμενο τύπου Gene(item) και ένα protected Node για να δείχνει στο επόμενο Node) για να κατασκευάσουμε τους κόμβους (παρόμοια κατασκευή) για την κάθε δομή. Το μέγεθος της κάθε δομής, προκειμένου να είναι O(1) αρχικοποιείται σαν 0, και μεταβάλλεται καθώς προστίθενται ή αφαιρούνται στοιχεία από τις δομές (βλ. παρακάτω). Στις μεθόδους που αδυνατούν να υλοποιηθούν αν οι δομές είναι άδειες έχει προστεθεί εξαίρεση NoSuchElementException, σύμφωνα με την εκφώνηση. Τέλος υπάρχουν οι μέθοδοι printStack και printQueue οι οποίες υλοποιήθηκαν σύμφωνα με την εκφώνηση, και τυπώνουν τα περιεχόμενα των δομών.

Στοίβα(StringStackImpl)

- public void push(Gene data){…} Δέχεται ένα όρισμα αντικειμένου Gene, δημιουργεί έναν καινούριο κόμβο στην κορυφή της στοίβας για να το αποθηκεύσει και το συνδέει με το $2^{ο}$, μέσω της κλάσης Node, ενώ αυξάνει την τιμή μεγέθους της στοίβας.
- public Gene pop() throws NoSuchElementException{..} Επιστρέφει σαν αντικείμενο Gene το περιεχόμενο του κορυφαίου κόμβου,τον αφαιρεί από την στοίβα και μειώνει την τιμή του μεγέθους της.

- public Gene peek() throws NoSuchElementException{…} Επιστρέφει σαν αντικείμενο Gene την πληροφορία του κορυφαίου κόμβου, δίχως να τον διαγράφει ή να μεταβάλει το μέγεθος της στοίβας.

Ουρά(IntQueueImpl)

- public void put(Gene data){…} Δέχεται ένα όρισμα αντικειμένου Gene, δημιουργεί έναν καινούριο κόμβο στο πίσω μέρος της ουράς για να το αποθηκεύσει και συνδέει το προηγούμενο με αυτό, μέσω της κλάσης Node, ενώ αυξάνει την τιμή μεγέθους της ουράς.
- Public Gene get() throws NoSuchElementException{…} Επιστρέφει σαν αντικείμενο Gene το περιεχόμενο του πρώτου κόμβου, τον αφαιρεί από την ουρά και μειώνει την τιμή του μεγέθους της.
- public Gene peek() throws NoSuchElementException{…} Επιστρέφει σαν αντικείμενο Gene την πληροφορία του 1^{ου} κόμβου, δίχως να τον διαγράφει ή να μεταβάλει το μέγεθος της στοίβας.
- Για τους σκοπούς του μέρους Γ υλοποιήσαμε μια μέθοδο η οποία αλλάζει το αντικείμενο του πρώτου κόμβου (συγκεκριμένα για το μέρος γ αλλάζει τον αριθμό των διαθέσιμων μετοχών), την public void setFirstData() throws NoSuchElementException{…}

Μέρος β) TagMachine

Σε αυτό το πρόγραμμα εξετάζουμε αν οι HTML ετικέτες ταιριάζουν (δηλαδή όποια ανοίγει πρέπει και να κλείνει). Αρχίζουμε με την εύρεση του φακέλου από τον πίνακα String args[], καθώς το path του φακέλου έχει περαστεί ως παράμετρος του προγράμματος. Χρησιμοποιούμε αντικείμενο bufferedReader προκειμένου να διαβάσουμε το αρχείο γραμμή-γραμμή. Κάθε φορά που σε κάποια γραμμή εντοπιστεί εισαγωγικό ανοίγματος ή κλεισίματος μιας ετικέτας (‘<’/ ‘>’) η θέση του στην γραμμή αποθηκεύεται σε μια μεταβλητή. Όταν αποθηκευτεί η θέση του εισαγωγικού κλεισίματος, χρησιμοποιούμε μεθόδους substring, απομονώνουμε το κείμενο μεταξύ των εισαγωγικών, και με την χρήση της ReconTag αποφασίζουμε εάν το κείμενο ανοίγει ή κλείνει μια ετικέτα, εξετάζοντας αν ο πρώτος της χαρακτήρας είναι το ΄/΄. Μετά από αυτό χρησιμοποιούμε την στοίβα από το Μέρος Α: εάν μια ετικέτα ανοίγει, το κείμενό της προστίθεται στην στοίβα. Έτσι, το πρόγραμμά μας εστιάζει στις

εμφωλευμένες ετικέτες, οι οποίες πρέπει να κλείσουν και πρώτες. Είναι λογικό η πιο εσωτερική ετικέτα να κλείσει πρώτη. Επομένως, εάν συναντήσουμε μια ετικέτα κλεισίματος, εξετάζουμε αν το περιεχόμενό της (πλην του χαρακτήρα ‘/’) ταιριάζει με το κορυφαίο περιεχόμενο της στοίβας. Αν όντως ταιριάζει το στοιχείο διαγράφεται από την στοίβα, και το επόμενο περιμένει να κλείσει. Διαφορετικά, η στοίβα θα παραμείνει γεμάτη, μέχρι να τελειώσουν οι γραμμές του αρχείου. Τέλος, εξετάζουμε απλά αν η στοίβα είναι άδεια (δηλαδή αν όλες η ετικέτες έχουν κλείσει) και τυπώνουμε την απάντησή μας.

Μέρος Γ

Η 3^{η} άσκηση της εργασίας στηρίζεται στην ανάγνωση ενός αρχείου txt και την εξαγωγή πληροφοριών από αυτό. Το πρόγραμμα ξεκινάει με το άνοιγμα του αρχείου περνώντας στο όρισμα args[] την τοποθεσία του αρχείου που θα διαβαστεί. Αρχικά, έχοντας ανοίξει το αρχείο μας χρησιμοποιώντας BufferedReader, εντοπίζουμε το εξής: κάθε γραμμή του αρχείου δηλώνει την αγορά ή την πώληση μετοχών, οπότε έχουμε 2 τρόπους χειρισμού της κάθε γραμμής (θεωρώντας πως το δοσμένο κείμενο θα βρίσκεται στην κατάλληλη για ανάγνωση μορφή). Στη συνέχεια, με τη βοήθεια των regular expressions θα καταφέρουμε να απομονώσουμε από κάθε γραμμή του κειμένου μονάχα τους αριθμούς, διότι μονάχα αυτοί μας ενδιαφέρουν και ανάλογα με το αν η γραμμή αρχίζει με buy ή sell, εκμεταλλευόμαστε με διαφορετικό τρόπο τους αριθμούς αυτούς.

Η κύρια διαδικασία μας (υλοποιήσαμε όλο το πρόγραμμα μέσα στη main, δεν χρησιμοποιήσαμε συναρτήσεις άλλες εντός της κλάσης), ξεκινά με το κεντρικό while το οποίο διαβάζει κάθε σειρά του αρχείου και την οποία επεξεργαζόμαστε για να αντλήσουμε τα κατάλληλα δεδομένα που χρειαζόμαστε. Όπως υποδείξαμε πριν, κάθε γραμμή του αρχείου αρχίζει με buy ή sell.

Περίπτωση buy:

Κάθε γραμμή που αρχίζει με buy περιέχει 2 αριθμούς, γνωρίζουμε πως αυτοί οι αριθμοί είναι: η ποσότητα των μετοχών που αγοράσαμε και η αντίστοιχη τιμή πώλησης αυτών. Έτσι, με την χρήση της μεταβλητής i (βλέπε

εξήγηση παρακάτω) , μπορούμε πλέον να τοποθετούμε κατάλληλα σε κάθε ουρά τα δεδομένα μας (χρησιμοποιούμε 2 παράλληλες ουρές, μία που περιέχει την ποσότητα των μετοχών, μία για την τιμή αγοράς αυτών των μετοχών).

Περίπτωση sell:

Σε αυτήν την περίπτωση, αποθηκεύουμε σε προσωρινές μεταβλητές(sellQuantity, sellPrice) την ποσότητα των μετοχών που θέλουμε να πουλήσουμε και την τιμή πώλησης τους. Κάνουμε έναν έλεγχο μόλις περαστούν οι τιμές για την περίπτωση που δεν μας φτάνουν οι μετοχές να πουλήσουμε, όπου το πρόγραμμα θα ειδοποιήσει τον χρήστη για το πρόβλημα και θα τερματιστεί. Αν το προηγούμενο δεν ισχύει, τότε, αφαιρεί τις απαιτούμενες μετοχές από τις παλαιότερες στις νεότερες και προσθέτει στην μεταβλητή profit τον αριθμό → αριθμός μετοχών*(κόστος πώλησης μετοχών – κόστος αγοράς μετοχών), το οποίο αν είναι θετικό σηματοδοτεί κέρδος, αλλιώς ζημία. Η διαδικασία αυτή συνεχίζεται έως ότου ο αριθμός των υπολειπόμενων προς πώληση μετοχών είναι μικρότερος από τις διαθέσιμες μετοχές του πλέον πρώτου κόμβου που είναι στην ουρά.

Διευκρίνιση i :

Η μεταβλητή i είναι ένας ακέραιος αριθμός ο οποίος μας βοηθάει στην διάκριση των αριθμών και στην τοποθέτηση στην σωστή τους στοίβα. Παρατηρούμε ότι ο $1^{ος}$ αριθμός σε μια σειρά buy θα είναι ο αριθμός των μετοχών και ο $2^{ος}$ θα είναι η τιμή απόκτησής τους. Έτσι, βλέπουμε ότι ανά 2 αριθμούς τοποθετούμε τον $1^{ο}$ στην στοίβα μετοχών.

Η ομάδα μας

Γεώργιος-Στέφανος Μεϊδάνης, p3170107, giorgosmeid@gmail.com

Κωνσταντίνος Κωνσταντόπουλος , p3170086, p3170086@dias.aueb.gr