1η Εργασία

ΕΠΑΜΕΙΝΩΝΔΑΣ ΙΩΑΝΝΟΥ(3140059)

ΑΛΕΞΑΝΔΡΟΣ ΚΟΝΟΜΗΣ(3140085)

ΑΝΑΦΟΡΑ ΠΑΡΑΔΟΣΗΣ

Η συγκεκριμένη εργασία αποτελείται από 3 μέρη. Το πρώτο και το τρίτο μέρος αποτελούν υλοποιήσεις Αφηρημένων Τύπων Δεδομένων όπως η στοίβα και η ουρά FIFO , ενώ το δεύτερο μέρος είναι εφαρμογή που βασίζεται στην ιδέα και υλοποίηση του ΑΤΔ της στοίβας που πραγματοποιήθηκε στο πρώτο μέρος. Στην εργασία έγινε χρήση Generics οπότε τα 2 interfaces έχουν προσαρμοστεί καταλλήλως. Επίσης έχει δημιουργηθεί μία κλάση Node η οποία χρησιμοποιείται από όλες τις υλοποιήσεις. Παρακάτω παρουσιάζονται αναλυτικά ο τρόπος σκέψης και οι ιδέες για την επιλογή του κώδικα ξεχωριστά για το κάθε μέρος.

Μέρος Α: Όσον αφορά την υλοποίηση της στοίβας κατασκευάσαμε μια κλάση που ονομάζεται StringStackImpl σύμφωνα με τις οδηγίες της εκφώνησης η οποία κάνει implement το αντίστοιχο Interface που έχει δοθεί(StringStack), το οποίο περιέχει τις μεθόδους που θα πρέπει να υλοποιήσουμε. Αρχικά δηλώνεται μια μεταβλητή αντικειμένου Node η οποία ονομάζεται top και θα αναπαριστά την κορυφή της στοίβας(στην αρχή είναι null προφανώς). Επίσης αρχικοποιούμε και μια μεταβλητή int size, η οποία θα αναπαριστά το μέγεθος της στοίβας και στην αρχή ισούται με 0. Στη συνέχεια υπάρχει ένας κατασκευαστής StringStackImpl() ο οποίος είναι κενός καθώς το μόνο που μας ενδιαφέρει είναι να κατασκευαστεί ένα αντικείμενο StringStackImpl αρχικά. Η μέθοδος boolean isEmpty() επιστρέφει την τιμή top==null; , καθώς αν η στοίβα είναι άδεια τότε ο κόμβος top(Node<T> top) είναι null . Οπότε αν η στοίβα είναι άδεια επιστρέφει true, αλλιώς false. Η μέθοδος void push(T item) είναι μια βασική λειτουργία της στοίβας ως ΑΤΔ και προσθέτει στην κορυφή της ένα στοιχείο(item) τύπου Τ. Γι'αυτό το λόγο κατασκευάζουμε ένα νέο κόμβο Node<T> node=new Node<>(item); που περιέχει το συγκεκριμένο στοιχείο. Έπειτα αυξάνουμε τη μεταβλητή size κατά 1, καθώς με τη λειτουργία push αυξάνεται κατά ένα το μέγεθος της στοίβας. Μετά με τη συνθήκη, αν δεν είναι άδεια η στοίβα κάνει την αναφορά του κόμβου που δημιουργήσαμε με το συγκεκριμένο στοιχείο node.next=top, δηλαδή είναι σα να συνδέουμε το συγκεκριμένο κόμβο με την κορυφή της λίστας. Έπειτα αλλάζουμε την κορυφή με το να δείχνει πλέον η μεταβλητή στο νέο κόμβο(top=node;). Η μέθοδος pop αφαιρεί και επιστρέφει τον κόμβο από την κορυφή της στοίβας. Αρχικά ελέγχουμε αν η στοίβα είναι άδεια χρησιμοποιώντας τη μέθοδο isEmpty() και αν είναι πετάει εξαίρεση τύπου NosuchElementException και τυπώνει Stack is Empty. Διαφορετικά, δίνουμε σε έναν

προσωρινό κόμβο (που ονομάζεται previous_top) την τιμή του κόμβου top ώστε να κρατήσουμε την τιμή που περιείχε η προηγούμενη κορυφή(top) της στοίβας. Έπειτα κάνομε τη μεταβλητή top=top.next ώστε πλέον ο κατώτερος κόμβος να αποτελεί τη νέα κορυφή της στοίβας. Για λόγους σύμβασης, έχουμε και το previous_top.next=null ώστε να αφαιρεθεί τελείως η σύνδεση της προηγούμενης κορυφής με τη στοίβα. Το size—μειώνει τη μεταβλητή size κατά ένα καθώς αφαιρείται ένα στοιχείο απ τη στοίβα. Τέλος, επιστρέφουμε το στοιχείο του κόμβου που αποτελούσε την προηγούμενη κορυφή της στοίβας(top) με την εντολή return previous_top.item. Η μέθοδος peek επιστρέφει το στοιχείο της κορυφής της στοίβας χωρίς όμως να το αφαιρεί απ τη στοίβα. Έτσι, με παρόμοιο τρόπο αν η στοίβα είναι άδεια πετάει εξαίρεση και τυπώνει το ίδιο μήνυμα με τη μέθοδο pop. Επιστρέφει top.item σε διαφορετική περίπτωση, δηλαδή το στοιχείο του κόμβου της κορυφής που επιθυμούμε. Η μέθοδος printStack τυπώνει τα στοιχεία των κόμβων που βρίσκονται στη στοίβα. Πάλι αν είναι άδεια η στοίβα, τυπώνουμε αντίστοιχο μήνυμα, ενώ σε διαφορετική περίπτωση χρησιμοποιούμε την εντολή Node<T> node=top για να ξεκινήσουμε απ την κορυφή της στοίβας και να αρχίσουμε να τυπώνουμε μέχρι να φτάσουμε στο πρώτο στοιχείο της στοίβας με τη συνθήκη while(node!=null) και μέσα στη while κάνουμε το node=node.next, δηλαδή προχωράμε στους κόμβους και τους εκτυπώνουμε κατά έναν ξεκινώντας απ την κορυφή μέχρι να φτάσουμε στο τέλος. Η μέθοδος size επιστρέφει τη μεταβλητή size που έχει την τιμή του παρόντος πλήθους της στοίβας. Σχετικά με την υλοποίηση της ουράς έχουμε την κλάση StringQueueImpl που κάνει Implement το Interface StringQueue. Αρχικά έχουμε τους κόμβους Node<T> head και Node<T> tail για την αρχή της ουράς και το τέλος της αντίστοιχα. Επίσης έχουμε και μια μεταβλητή size που χρησιμοποιείται και δω για το μέγεθος της ουράς. Η μέθοδος isEmpty() επιστρέφει head==null δηλαδή είναι άδεια αν δεν έχει αρχικοποιηθεί ακόμα ο κόμβος head. Η μέθοδος put βάζει τον κόμβο με το συγκεκριμένο στοιχείο στο τέλος της ουράς. Αν η ουρά είναι άδεια τότε το head θα δείχνει στο νέο κόμβο που τοποθετούμε αλλιώς έχουμε την εντολή tail.next=node; για να συνδέσουμε τον τελευταίο κόμβο της ουράς με το νέο κόμβο(η size αυξάνεται κατά ένα). Μετά το tail πλέον θα δείχνει στο νέο κόμβο καθώς είναι το τελικό στοιχείο. Η μέθοδος get πετάει εξαίρεση και τυπώνει αντίστοιχο μήνυμα αν η ουρά είναι άδεια, διαφορετικά έχουμε ένα προσωρινό κόμβο Node<T> oldhead=head για να γυρίσουμε το στοιχείο του αρχαιότερου κόμβου. Μετά η μεταβλητή θα δείχνει στο δεύτερο αρχαιότερο στοιχείο με την εντολή head=head.next, καθώς αποκόπτουμε και τη σύνδεση του παλιού head με oldhead.next=null;(Το size προφανώς μειώνεται κατά ένα).Επιστρέφουμε oldhead.item. Η μέθοδος peek πετά εξαίρεση αν η ουρά είναι άδεια, αλλιώς επιστρέφει head.item. Η printQueue λειτουργεί με τον τρόπο που περιγράφηκε παραπάνω δηλαδή τυπώνει τα στοιχεία κάθε κόμβου της ουράς ξεκινώντας απ το αρχαιότερο και φτάνοντας στο πιο πρόσφατο. Η size επιστρέφει το πλήθος της ουράς.

Μέρος Γ: Σχετικά με την υλοποίηση του Interface της ουράς με ένα δείκτη έχουμε την κλάση StringQueueWithOnePointer που κάνει implement το Interface της ουράς. Αρχικά όπως το όνομα δηλώνει, χρησιμοποιείται μόνο ένας δείκτης αντί για δύο(head και tail). Δηλώνουμε στην αρχή τον κόμβο Node<T> last που θα είναι ο μοναδικός μας δείκτης για την υλοποίηση, καθώς και μία ακέραια μεταβλητή size που θα αναπαριστά το μέγεθος της ουράς. Η μέθοδος isEmpty επιστρέφει last==null καθώς αν η ουρά είναι άδεια ο δείκτης last θα είναι null. Η μέθοδος put τοποθετεί ένα στοιχείο στην ουρά αλλά λόγω του ότι γίνεται μόνο με ένα δείκτη η διαδικασία είναι λίγο πιο περίπλοκη. Για να επιτύχουμε τη χρήση ενός μόνο δείκτη χρησιμοποιούμε κυκλική λίστα αντί για λίστα μονής σύνδεσης. Αυτό σημαίνει ότι ο τελευταίος κόμβος θα συνδέεται με τον πρώτο και αντί να έχει την αναφορά του Null η αναφορά του (next) θα δείχνει στο πρώτο στοιχείο. Οπότε με αυτή τη λογική στην εισαγωγή του πρώτου στοιχείου με τη μέθοδο put δηλαδή όταν η ουρά είναι άδεια ο δείκτης last δείχνει στο νέο κόμβο με last=node, απ τον οποίο θέλουμε να εισάγουμε το στοιχείο, αλλά ταυτόχρονα έχουμε και την εντολή last.next=node καθώς η λίστα είναι κυκλική. Στην περίπτωση της μονής λίστας το συγκεκριμένο θα ήταν null. Οπότε όταν εισάγουμε ένα δεύτερο στοιχείο «συνδέουμε» το νέο κόμβο με τον πρώτο ως εξής node.next=last.next. Με αυτό τον τρόπο ο νέος κόμβος δείχνει στην αρχή. Πρέπει να αλλάξουμε επίσης και το που δείχνει η μεταβλητή last καθώς δείχνει ακόμα στην αρχή της κυκλικής λίστας(αυτό γίνεται με την εντολή last.next=node) και τέλος γίνεται ο νέος κόμβος το τελευταίο στοιχείο με last=node και έχουμε επίσης την αύξηση του μεγέθους size++. Εν ολίγοις «συνδέουμε» το νέο κόμβο με τον τελευταίο και τον πρώτο κόμβο και έπειτα το μετατρέπουμε σε last το ίδιο ακολουθώντας ακριβώς την ίδια διαδικασία σε επόμενη εισαγωγή. Η μέθοδος get πετά εξαίρεση αν η ουρά είναι άδεια και τυπώνει μήνυμα. Σε αντίθετη περίπτωση, κάνουμε έναν προσωρινό κόμβο που είναι έχει τη διεύθυνση του πρώτου στοιχείου με Node<T> node =last.next και έχουμε τις εξής επιλογές: α) Αν η ουρά έχει μόνο ένα στοιχείο τότε επειδή αφαιρείται η μεταβλητή last πλέον δε θα δείχνει πουθενά και έχουμε last=null και β) αν έχει πάνω από έναν κόμβο η ουρά να να συνδέσουμε τον κόμβο last στο δεύτερο αρχαιότερο κόμβο(last.next=node.next) καθώς το node.next δείχνει στο δεύτερο κόμβο αφού το node αποτελεί τον πρώτο κόμβο. Με αυτό τον τρόπο το node που ήταν το πρώτο στοιχείο στην κυκλική λίστα αποσυνδέεται και γι αυτό το λόγο μειώνουμε τη μεταβλητή size—κατά ένα και επιστρέφουμε το στοιχείο του node με return node.item. Η μέθοδος peek πετάει εξαίρεση αν η ουρά είναι άδεια, αλλιώς γυρνάει χωρίς να αφαιρεί το στοιχείο του πρώτου κόμβου με return last.next.item. Η μέθοδος printqueue τυπώνει τα στοιχεία των κόμβων από τον αρχαιότερο ως το νεότερο αρχικοποιώντας ένα κόμβο με τον πρώτο κόμβο της ουράς ως εξής Node<T> node=last.next. Μετά χρησιμοποιείται ένα απλό for loop. Τέλος η μέθοδος size επιστρέφει το μέγεθος της ουράς.

Μέρος Β: Στο μέρος Β γίνεται η υλοποίηση στην κλάση Thiseas και η εκτέλεση του κώδικα έγινε από command line. Αρχικά δημιουργήθηκε μια στατική κλάση που ονομάζεται Position η οποία θα αποθηκεύει κάθε φορά τις συντεταγμένες x και y. Η βασική λογική της υλοποίησης της εφαρμογής είναι όπως παρουσιάζεται παρακάτω. Αρχικά έχουμε 3 μεθόδους τη main, τη check_maze που ελέγχει για πιθανά λάθη στα δεδομένα εισόδου, δηλαδή στο text file και τη solve_maze που ψάχνει για έξοδο. Στην αρχή της main σώζουμε το path του text αρχείου σε ένα String με την εντολή String path=args[0] και μετά φτιάχνουμε αντικείμενα Scanner και BufferedReader(ο λόγος που χρησιμοποιούμε και bufferedreader είναι ότι στο διάβασμα γραμμών παρουσίαζε πρόβλημα το αντικείμενο Scanner) για να επεξεργαστούμε το κείμενο μέσα στην check_maze και να ελέγξουμε για πιθανά λάθη. Έπειτα αν η μορφή του text file είναι λανθασμένη η check_maze πετάει εξαίρεση και τυπώνει μήνυμα λάθους. Αν η μορφή του text document είναι σωστή αποθηκεύουμε τις γραμμές των διαστάσεων του πίνακα σε ακέραια μεταβλητή rows, τις στήλες των διαστάσεων του πίνακα σε ακέραια μεταβλητή columns και τη θέση του Ε σε ακέραια μεταβλητή e_x και e_y. Ύστερα αποθηκεύουμε σε ένα δυσδιάστατο πίνακα char buffer{}{]=new char[rows][columns],με 2 for loops τα στοιχεία του text εκτός από τις 4 ακέραιες μεταβλητές που είπαμε προηγουμένως. Έπειτα φτιάχνουμε ένα αντικείμενο Position που ονομάζεται E_pos που περιέχει τη θέση του Ε. Μετά καλούμε τη solve_maze με arguments το E_pos και τον πίνακα χαρακτήρων(char buffer{]{]) με σκοπό να βρούμε αν υπάρχει έξοδος στο λαβύρινθο. Τα βήματα που ακολουθούνται στη μέθοδο αυτή είναι : α) Φτιάξε ένα αντικείμενο StringStackImpl τύπου Position(δηλαδή στοίβα που θα περιέχει αντικείμενα με συντεταγμένες x και y),β) Εισαγωγή του πρώτου στοιχείου που είναι το E_pos(με τη μέθοδο και λειτουργία push). γ) Κίνηση δεξιά ή αριστερά ή πάνω ή κάτω(ανάλογα με το που υπάρχουν 0 και 1 που είναι το Ε αρχικά κλπ) και ανάλογα που κινούμαστε κάνουμε push το σημείο που πλέον είμαστε, στο stack, δ) Σε όποιο σημείο του πίνακα πηγαίνει το σημειώνουμε με συγκεκριμένο γράμμα εντός του πίνακα για αποφυγή επίσκεψης, ε) αν δεν μπορεί να γίνει καμία κίνηση γίνεται pop το τελευταίο στοιχείο της στοίβας και Backtracking έως κάπου που δεν είναι αδιέξοδο, ζ) μετά από κάθε κίνηση(όχι backtracking) ελέγχεται αν η στήλη ή γραμμή είναι πρώτη ή τελευταία και αν είναι τότε έχουμε φτάσει σε έξοδο και το πρόγραμμα την τυπώνει. Τέλος, αν δεν υπάρχει έξοδος ο αλγόριθμος ψάχνει όλα τα δυνατά μονοπάτια έως ότου φτάνει σε τελικό αδιέξοδο και μετά κάνει backtracking ως τη θέση του Ε καθώς δεν έχει που να πάει και τυπώνει ότι δεν υπάρχει έξοδος. Περισσότερες λεπτομέρειες σχετικά με την υλοποίηση του κώδικα υπάρχουν στα σχόλια του Thiseas.java.