Μαρία Σχοινάκη

Έχει πραγματοποιηθεί υλοποίηση με **generics** σε **όλα** τα μέρη της εργασίας.

Μέρος Α - StringStackImpl, StringQueueImpl

ΑΡΧΕΙΑ:

StringStackImpl.java, StringQueueImpl.java, Node.java, StringStack.java, StringQueue.java

Στην εργασία ξεκινήσαμε δημιουργώντας 2 κλάσεις, την **StringStackImpl** και την **StringQeueImpl** που υλοποιούν τις διεπαφές **StringStack** και **StringQueue** αντίστοιχα.

StringStackImpl

Αρχικοποιήσαμε μια μεταβλητή head τύπου Node που εκφράζει τον κόμβο του τελευταίου στοιχείου που εισήχθη στην Στοίβα, σε null και μια μεταβλητή size που μετράει το πλήθος των στοιχείων της στοίβας σε μια συγκεκριμένη χρονική στιγμή, σε 0 αφού η στοίβα αρχικά είναι άδεια. Υλοποιήσαμε την μέθοδο **isEmpty()**, που ελέγχει αν η στοίβα είναι άδεια(Αν το head είναι null τότε δεν υπάρχει κανένα στοιχείο στην στοίβα, αφού η μεταβλητή που εκφράζει το στοιχείο που είναι πάνω πάνω στην στοίβα είναι "το κενό"). Αμέσως μετά υλοποιήσαμε την **push()**, που εισάγει ένα νέο στοιχείο στην στοίβα με τις κατάλληλες ενέργειες(Αν η στοίβα δεν είναι άδεια πρέπει να θέσουμε το head της *χρονικής στιγμής 0* ως το 2^{ο} κάτω από το head της *χρονικής στιγμής 1* που είναι το στοιχείο εισαγωγής. Επίσης για λόγους υλοποίησης αυξάνουμε το size κατά 1 αφού προσθέθηκε ένα στοιχείο στην στοίβα). Ανάλογα έγινε και η υλοποίηση της μεθόδου **pop()** η οποία εξάγει ένα στοιχείο από την στοίβα και συγκεκριμένα το πάνω πάνω στοιχείο(Πρέπει προφανώς να ελεγχθεί αν η στοίβα είναι άδεια και αν ναι να εμφανίζεται κάποιο *exception*, καθώς δεν υπάρχει κάποιο στοιχείο για να εξαχθεί. Αλλιώς, αν υπάρχει τουλάχιστον 1 στοιχείο στην στοίβα η μεταβλητή head θα δείχνει πλέον στο επόμενο στοιχείο (head.getNext()) ή null αν δεν υπάρχει επόμενο στοιχείο. Επίσης για λόγους υλοποίησης μειώνουμε το size κατά 1 αφού αφαιρέθηκε ένα στοιχείο από την στοίβα). Υλοποιήσαμε μετά την μέθοδο **peek()** , η οποία επιστρέφει το πάνω πάνω στοιχείο της στοίβας χωρίς όμως να το εξάγει από την στοίβα.(Προφανώς ελέγχουμε ξανά άμα η στοίβα είναι άδεια γιατί αν είναι, δεν υπάρχει στοιχείο να επιστραφεί, οπότε δημιουργείται exception). Μετά υλοποιήσαμε την μέθοδο **printStack()** , η οποία εμφανίζει τα στοιχεία της στοίβας(Αρχικοποιούμε μια μεταβλητή temp τύπου Node σε head και έπειτα δημιουργούμε μια επανάληψη η οποία τρέχει όσο η temp δεν είναι null. Σε κάθε επανάληψη αφού εμφανιστεί το στοιχείο η temp γίνεται ο επόμενος κόμβος στην σειρά κοκ. Προφανώς άμα η στοίβα είναι άδεια η επανάληψη δεν θα τρέξει και αν δεν είναι, όταν εμφανιστεί το τελευταίο στοιχείο η temp θα γίνει null και θα βγει από την επανάληψη. Προφανώς χρησιμοποιούμε την temp για να μην χαθεί η τιμή του head). Τέλος, υλοποιήσαμε την μέθοδο **size()**, η οποία επιστρέφει το πλήθος των στοιχείων που περιέχει η στοίβα. Η υλοποίηση έγινε με **generics** ώστε να μπορεί η στοίβα να αποτελείται από οποιουδήποτε τύπου αντικείμενα. Ο τύπος καθορίζεται όταν χρησιμοποιείται η στοίβα, δηλαδή στην εργασία μας στο Β μέρος. Οι μέθοδοι size(),push() και pop() υλοποιούνται σε O(1) χρόνο.

Μαρία Σχοινάκη

StringQueueImpl

Με ακριβώς ανάλογο τρόπο υλοποιήθηκε και η ουρά. Η διαφορά σε αυτήν την δομή δεδομένων είναι ότι υπάρχουν 2 μεταβλητές, η head και η tail, τύπου Node. Η head εκφράζει τον κόμβο του οποίου το στοιχείο θα εξαχθεί πρώτο, αν η ουρά δεν είναι άδεια, ενώ η tail εκφράζει τον κόμβο που περιέχει το πιο πρόσφατο στοιχείο που εισήχθη στην ουρά. Ξεκινάμε αρχικοποιώντας τες με null, και με 0 μια μεταβλητή size που μετράει το πλήθος των στοιχείων της ουράς σε μια συγκεκριμένη χρονική στιγμή, αφού η ουρά αρχικά είναι άδεια. Υλοποιήσαμε την μέθοδο **isEmpty()**, που ελέγχει αν η ουρά είναι άδεια(Αν το head είναι null τότε δεν υπάρχει κανένα στοιχείο στην ουρά, αφού η μεταβλητή που εκφράζει το στοιχείο που είναι πρώτο προς εξαγωγή στην ουρά, είναι "το κενό"). Αμέσως μετά υλοποιήσαμε την μέθοδο **put()**, η οποία εισάγει ένα στοιχείο στην ουρά(Ελέγχουμε αν η ουρά είναι άδεια, γιατί αν όντως είναι, το head και το tail δείχνουν null, οπότε με την εισαγωγή του $1^{ου}$ στοιχείου στην ουρά, πρέπει και οι 2 μεταβλητές να δείχνουν τον ίδιο κόμβο που θα περιέχει το στοιχείο εισαγωγής. Αν τώρα η ουρά δεν είναι άδεια, τότε απλά αλλάζει το tail και δείχνει πλέον σε έναν κόμβο που περιέχει το στοιχείο που εισήχθη. Επίσης για λόγους υλοποίησης αυξάνουμε το size κατά 1 αφού προσθέθηκε ένα στοιχείο στην ουρά). Σειρά έχει η υλοποίηση την μεθόδου **get()**, η οποία εξάγει ένα στοιχείο από την ουρά και συγκεκριμένα το στοιχείο που περιέχεται στον head(Πρέπει προφανώς να ελεγχθεί αν η ουρά είναι άδεια και αν ναι να εμφανίζεται κάποιο *exception*, καθώς δεν υπάρχει κάποιο στοιχείο για να εξαχθεί. Αλλιώς, αν υπάρχει 1 μόνο στοιχείο στην ουρά να γίνονται οι μεταβλητές head και tail, null δηλαδή να αδειάσει η ουρά, ενώ σε κάθε άλλη περίπτωση το head πρέπει να "διαγράφεται" και πλέον να δείχνει στον κόμβο που περιέχει το αμέσως επόμενο στοιχείο για εξαγωγή. Επίσης για λόγους υλοποίησης μειώνουμε το size κατά 1 αφού αφαιρέθηκε ένα στοιχείο από την ουρά). Υλοποιήσαμε μετά την μέθοδο **peek()** , η οποία επιστρέφει το πρώτο προς εξαγωγή στοιχείο της ουράς χωρίς όμως να το εξάγει από την ουρά.(Προφανώς ελέγχουμε ξανά άμα η ουρά είναι άδεια γιατί αν είναι, δεν υπάρχει στοιχείο να επιστραφεί, οπότε δημιουργείται exception). Μετά υλοποιήσαμε την μέθοδο **printStack()** , η οποία εμφανίζει τα στοιχεία της ουράς(Αρχικοποιούμε μια μεταβλητή tem τύπου Node σε head και έπειτα δημιουργούμε μια επανάληψη η οποία τρέχει όσο η tem δεν είναι null. Σε κάθε επανάληψη αφού εμφανιστεί το στοιχείο η tem γίνεται ο επόμενος κόμβος στην σειρά κοκ. Προφανώς άμα η ουρά είναι άδεια η επανάληψη δεν θα τρέξει και αν δεν είναι, όταν εμφανιστεί το τελευταίο στοιχείο η tem θα γίνει null και θα βγει από την επανάληψη. Προφανώς χρησιμοποιούμε την tem για να μην χαθεί η τιμή του head). Τέλος, υλοποιήσαμε την μέθοδο **size()**, η οποία επιστρέφει το πλήθος των στοιχείων που περιέχει η ουρά. Η υλοποίηση έγινε με **generics** ώστε να μπορεί η ουρά να αποτελείται από οποιουδήποτε τύπου αντικείμενα. Ο τύπος καθορίζεται όταν χρησιμοποιείται η ουρά. Οι μέθοδοι size(),put() και get() υλοποιούνται σε O(1) χρόνο.

Μαρία Σχοινάκη

<u>Μέρος Β – Thiseas</u>

ΑΡΧΕΙΑ:

Thiseas.java, StringStackImpl.java, StringStack.java, Node.java, Point.java

Σε αυτό το μέρος της εργασίας, χρησιμοποιώντας την υλοποίηση της στοίβας του μέρους Α φτιάξαμε ένα πρόγραμμα που διαβάζει ένα αρχείο(file), το οποίο δίνει το μέγεθος του πίνακα, τον πίνακα και τις συντεταγμένες της είσοδο, φτιάχνει έναν πίνακα MN διαστάσεων και τον διατρέχει μέχρι(αν) να βρει έξοδο. Η βασική ιδέα πίσω από το πρόγραμμα είναι η χρησιμοποίηση της στοίβας. Δεδομένου του πίνακα που διαβάζεται, και ότι το 0 σημαίνει πέρασμα ενώ το 1 όχι, δίναμε εντολή στο πρόγραμμα να εξετάζει τις 4 κατευθύνσεις πάνω, κάτω, δεξιά και αριστερά και κάθε φορά να μετακινείται στο πρώτο σημείο που εξετάζει και έχει μηδενικό. Κάθε φορά που έβρισκε τέτοιο σημείο, το έβαζε στην στοίβα με εισαγωγή και το μετέτρεπε στον πίνακα από 0 σε 10. Ο λόγος είναι για να μπορούμε να ελέγξουμε αν το συγκεκριμένο σημείο υπάρχει ήδη στην στοίβα (Δηλαδή ότι έχει περάσει ήδη από αυτό). Έτσι λοιπόν αν φτάσουμε σε ένα σημείο που πάνω κάτω και αριστερά έχει 1 ενώ δεξιά το 10 σημαίνει ότι ήρθε από δεξιά και είναι αδιέξοδο, άρα το εξάγουμε από την στοίβα και δεν ξαναπάμε σε αυτό το σημείο. Έτσι λοιπόν, η στοίβα περιέχει τις συντεταγμένες του εκάστοτε μονοπατιού που εξετάζεται κάθε φορά. Προφανώς, για να εξεταστεί η μη λειτουργικότητα ενός μονοπατιού, ξεκινάμε από το τελευταίο σημείο του και πάμε προς τα πίσω(backtracking), ελέγχοντας αν υπάρχει άλλο μονοπάτι όσο πάμε προς τα πίσω στην είσοδο. Αν τώρα φτάσει σε ένα από τα 4 άκρα του πίνακα(αριστερότερη, δεξιότερη στήλη ή πρώτη, τελευταία γραμμή και υπάρξει μονοπάτι που οδηγεί εκεί και υπάρχει στα άκρα μηδενικό, τότε το πρόγραμμα τερματίζεται και εμφανίζονται οι συντεταγμένες της εξόδου. Αν υπάρχει μηδενικό στα άκρα και δεν υπάρχει μονοπάτι προς αυτό, τότε τερματίζει και εμφανίζεται αντίστοιχο μήνυμα, ενώ αν δεν υπάρχει καν μηδενικό στα άκρα, τερματίζει ξανά και εμφανίζει κατάλληλο μήνυμα. Στο πρόγραμμα πρέπει να γίνουν και απαραίτητοι έλεγχοι, ώστε άμα η είσοδος βρίσκεται $1^{η}$ γραμμή να μην μπορεί να μετακινηθεί η αναζήτηση προς τα πάνω(δηλαδή να μην βγει έξω από τα όρια του πίνακα). Τέλος χρησιμοποιήσαμε μια καινούρια κλάση Point η οποία δημιουργεί αντικείμενα συντεταγμένων και βάζει κάθε αντικείμενο που η αναζήτηση χρειάζεται, στην στοίβα. Εφόσον έχει υλοποιηθεί με generics η στοίβα κάθε τύπος είναι αποδεκτός. Επιπλέον γίνεται έλεγχος για το αν υπάρχουν λανθασμένα δεδομένα στο .txt αρχείο και εμφανίζεται αντίστοιχο μήνυμα. (π.χ. αν οι γραμμές και οι στήλες που δίνονται δεν ταιριάζουν με αυτές του πίνακα που διαβάζεται ,αρνητικά όρια πίνακα, περισσότερες από μια είσοδοι κλπ.)

Μαρία Σχοινάκη

Μέρος Γ – StringQueueWithOnePointer

ΑΡΧΕΙΑ:

StringQueueWithOnePointer.java , StringQueue.java , Node.java

Σε συνέχεια του ερωτήματος Α δημιουργήσαμε μια κλάση StringQueueWithOnePointer η οποία υλοποιεί την διεπαφή StringQueue αλλά χρησιμοποιούμε έναν δείκτη αντί για 2 που έχει εξορισμού μία ουρά. Αρχικοποιήσαμε μια μεταβλητή tail τύπου Node σε null και μια μεταβλητή size, που μετράει το πλήθος των στοιχείων της ουράς σε μια συγκεκριμένη χρονική στιγμή, σε 0, αφού η ουρά αρχικά είναι άδεια. Η ιδέα πίσω από την χρήση μίας μόνο μεταβλητής, είναι ότι ο επόμενος κόμβος μετά τον tail(τελευταίο προς εξαγωγή), είναι ο 1^{ος} κόμβος (κυκλική λίστα). Άρα στην πραγματικότητα μπορούμε οποιαδήποτε στιγμή να επικαλεστούμε την head (δηλαδή το πρώτο στοιχείο της ουράς) με την tail.getNext(), αφού ο επόμενος κόμβος πρακτικά μετά τον tail είναι ο head (κυκλικά). Υλοποιήσαμε την μέθοδο **isEmpty()**, που ελέγχει αν η ουρά είναι άδεια(Αν το tail είναι null τότε δεν υπάρχει κανένα στοιχείο στην ουρά, αφού η μεταβλητή που εκφράζει το στοιχείο που είναι τελευταίο προς εξαγωγή στην ουρά, είναι "το κενό", δηλαδή δεν υπάρχει καν στοιχείο που μπορεί να εξαχθεί. Άρα στην ουσία η ουρά είναι άδεια). Αμέσως μετά υλοποιήσαμε την μέθοδο **put()**, η οποία εισάγει ένα στοιχείο στην ουρά(Ελέγχουμε αν η ουρά είναι άδεια, γιατί αν όντως είναι, το tail δείχνει null, οπότε με την εισαγωγή του 1^{ου} στοιχείου στην ουρά, πρέπει το tail πλέον να δείχνει στον κόμβο που περιέχει το στοιχείο εισαγωγής και το tail.getNext() να δείχνει τον εαυτό του (tail) . Αν τώρα η ουρά δεν είναι άδεια, τότε απλά το παλιό tail δείχνει στον καινούργιο κόμβο που εισήχθη (καινούργιο tail), ο οποίος με τη σειρά του δείχνει στο πρώτο στοιχείο της ουράς (κυκλική λίστα) . Επίσης για λόγους υλοποίησης αυξάνουμε το size κατά 1 αφού προσθέθηκε ένα στοιχείο στην ουρά). Σειρά έχει η υλοποίηση την μεθόδου **get()**, η οποία εξάγει ένα στοιχείο από την ουρά και συγκεκριμένα το στοιχείο που περιέχεται στον head δηλαδή στο tail.getNext() (Πρέπει προφανώς να ελεγχθεί αν η ουρά είναι άδεια και αν ναι να εμφανίζεται κάποιο *exception*, καθώς δεν υπάρχει κάποιο στοιχείο για να εξαχθεί. Αλλιώς, αν υπάρχει 1 μόνο στοιχείο στην ουρά να γίνεται η μεταβλητή tail, null δηλαδή να αδειάσει η ουρά, ενώ σε κάθε άλλη περίπτωση το head, δηλαδή το tail.getNext(), πρέπει να "διαγράφεται" και το tail.getNext() πλέον να δείχνει στον κόμβο που περιέχει το αμέσως επόμενο στοιχείο για εξαγωγή (καινούργιο head). Επίσης για λόγους υλοποίησης μειώνουμε το size κατά 1 αφού αφαιρέθηκε ένα στοιχείο από την ουρά). Υλοποιήσαμε μετά την μέθοδο **peek()** , η οποία επιστρέφει το πρώτο προς εξαγωγή στοιχείο της ουράς χωρίς όμως να το εξάγει από την ουρά.(Προφανώς ελέγχουμε ξανά άμα η ουρά είναι άδεια γιατί αν είναι, δεν υπάρχει στοιχείο να επιστραφεί, οπότε δημιουργείται exception, διαφορετικά επιστρέφονται τα δεδομένα του πρώτου στοιχείου, δηλαδή του tail.getNext()). Μετά υλοποιήσαμε την μέθοδο **printStack()** , η οποία εμφανίζει τα στοιχεία της ουράς(Αρχικοποιούμε μια μεταβλητή tem τύπου Node σε head, δηλαδή σε tail.getNext() αν και μόνο αν η ουρά έχει ένα τουλάχιστον στοιχείο, γιατί αλλιώς δεν μπορεί να εκτελεστεί η εντολή tail.getNext(). Μετά, τρέχουμε μια επανάληψη η οποία εμφανίζει τα

στοιχεία της ουράς ανά επανάληψη και κάνει την tem να δείχνει σε κάθε επόμενο κόμβο όσο προφανώς η tem είναι διάφορη του 1^{ου} στοιχείου, δηλαδή όσο προσπελαύνουμε την ουρά χωρίς να ξεπεράσουμε τον επόμενο κόμβο του tail, που είναι ο head(tail.getNext()),το οποίο θα είχε ως αποτέλεσμα τη συνεχή εμφάνιση των στοιχείων ατέρμονα. Τέλος, υλοποιήσαμε την μέθοδο **size()**, η οποία επιστρέφει το πλήθος των στοιχείων που περιέχει η ουρά. Η υλοποίηση έγινε με **generics** ώστε να μπορεί η ουρά να αποτελείται από οποιουδήποτε τύπου αντικείμενα. Ο τύπος καθορίζεται όταν χρησιμοποιείται η ουρά.