Όνομα: Αναστάσιος Παπαπαναγιώτου
ΑΜ: 3200143

Ερώτημα Α

IntQueue

Για να υλοποιήσω με Generic πρέπει να προσθέσω το απαραίτητο T στο IntQueueImpl.java και IntQueue.java και μετά χρησιμοποιώ αυτό αντί για Integer, String, κλπ. Καταρχάς, η ουρά πρέπει να έχει δύο δείκτες ένα head και ένα tail τα οποία θα είναι global μεταβλητές όπως και η μεταβλητή length η οποία κρατάει το μήκος της ουράς. Η υλοποίηση της isEmpty() είναι true αν το μήκος της ουράς είναι 0. Στην ουρά προσθέτω στοιχείο στο tail επειδή το head το χρειαζόμαστε για να αφαιρούμε στοιχεία σε O(1). Έτσι, ελέγχω αν είναι κενή ουρά και θέτω το head=tail=node γιατί θα υπάρχει ένα στοιχείο. Αν δεν είναι κενή, τότε ο δείκτης tail θα δείχνει στο καινούργιο Node που φτιάξαμε και επειδή το προσθέσαμε στο τέλος, το καινούργιο tail πρέπει να είναι το καινούργιο Node και τέλος αυξάνουμε το length κατά 1. Αφαιρούμε στοιχείο από την "αρχή" ή head της λίστας γιατί πρέπει να αφαιρέσουμε το πιο παλιό στοιχείο. Αν είναι κενή ουρά τότε θα υπάρξει υποχείλιση και πρέπει να μπεί exception. Αλλιώς, παίρνουμε την τιμή του πιο παλιού στοιχείου για να την επιστρέψουμε, και μετά ελέγχουμε αν το head.getNext() == null δηλαδή έχει μόνο ένα στοιχείο η ουρά και άρα στο τέλος η ουρά θα είναι κενή (head = tail = null) αλλιώς το head θα δείχνει στην επόμενη τιμή αρα προσπερνάει το πιο παλιό στοιχείο και δείχνει στο δεύτερο πιο παλιό, τέλος επιστρέφεται η τιμή και το length μειώνεται κατά 1. Το peek() θα επιστρέφει το πιο παλιό στοιχείο όμως αν είναι κενή ουρά πρέπει να κάνει throw exception. Για να κάνουμε print την ουρά ξεκινάμε από το head και προσθέτουμε σε ένα String το value του τωρινού Node, και τελειώνει μέχρι το currNode να φτάσει στο τέλος (null). Και το size() επιστρέφει την τιμή του length δηλαδή το μήκος της ουράς.

StringStack

Το isEmpty() είναι παρόμοιο με αυτό της IntQueue. Το push() προσθέτει στην κορυφή της στοίβας ένα στοιχείο, και ελέγχει πρώτα αν είναι κενή η στοίβα ώστε να θέσουμε το head (κορυφή) να είναι το καινούργιο node αλλιώς το head θα δείχνει στο καινούργιο node και αύξησε το ύψος της στοίβας κατά 1. Το pop() αφαιρεί το στοιχείο στην κορυφή της στοίβας με το να δείχνει στο επόμενο στοιχείο δηλαδή το δεύτερο πιο καινούργιο και ελέγχει πρώτα αν είναι κενή στοίβα και τέλος επιστρέφει την τιμή της κορυφής που αφαιρέσαμε και μείωσε το ύψος κατά 1. Το peek() επιστρέφει την τιμή της κορυφής της στοίβας. Το printStack() τυπώνει όλα τα στοιχεία της στοίβας ξεκινώντας από την κορυφή και μετά το τωρινό Node θα δείχνει στο επόμενο μέχρι να γίνει null. Τέλος το size() επιστρέφει το ύψος της στοίβας.

Ερώτημα Β

Για να δουλεύει σωστά πρέπει πρώτα να κλείνει το πιο καινούργιο tag και μετά τα επόμενα πιο καινούργια. Έτσι η χρήση της στοίβας είναι κατάλληλη. Έτσι αφού διαβαστεί το αρχείο, ελέγχουμε κάθε χαρακτήρα αν είναι ίσος με < τότε διάβασε το περιεχόμενο που υπάρχει

μέχρι να εμφανιστεί το > και παρέλειψε το / αν υπάρχει. Αν δεν υπάρχει το / μετά το < τότε προσθέτουμε το tag την κορυφή της στοίβας. Αν υπάρχει το / μετά το < τότε σημαίνει ότι κλείνει και ελέγχουμε αν το tag που κλείνουμε είναι το ίδιο με την κορυφή της στοίβας. Αν όχι σημαίνει ότι δεν κλείνει το σωστό και άρα το πρόγραμμα τελειώνει. Αν ναι, τότε αφαίρεσε το στοιχείο στην κορυφή και συνέχισε. Το πρόγραμμα τελειώνει όταν έχουμε φτάσει στο τέλος του αρχείου και στο τέλος πρέπει να ελέγξουμε αν υπάρχουν tags στη στοίβα δηλαδή αν υπάρχουν tags που δεν έχουν κλείσει.

Ερώτημα Γ

Όταν πωλούνται οι μετοχές πρώτου πωλούνται οι πιο παλιές και μετά όταν πουληθούν όλες τότε συνεχίζουμε στις επόμενες πιο παλιές. Για αυτό χρησιμοποιείται η ουρά. Αρχικά, διαβάζεται το αρχείο σειρά σειρά και αν ξεκινάει με το buy τότε διαβάζουμε το Pair δηλαδή την ποσότητα και την τιμή και την αποθηκεύουμε σε ένα Pair. Το Pair είναι μια κλάση στην οποία αποθηκεύεται η τιμή και η ποσότητα ενός trade είτε buy είτε sell. Και αυτό το pair μετά αποθηκεύεται στην ουρά. Αν κάνουμε sell τότε όσο το sellAmount δηλαδή πόσες μετοχές πουλάμε είναι μεγαλύτερο του 0, ελέγχουμε αν η ουρά είναι κενή δηλαδή δεν έχουμε αγοράσει κάποια μετοχή και είναι μεγαλύτερο του ποσού που έχει απομείνει από παλιότερες πωλήσεις δηλαδή αν πουλήσω μια μετοχή και μας μείνουν από αυτές που αγοράσαμε τότε η ουρά θα είναι κενή αλλά έχουμε από πριν που δεν έχουμε πουλήσει. Αν έχουμε μετοχές που έχουν μείνει από πριν και είναι περισσότερες από αυτές που πουλάμε τότε θα πουλήσουμε αυτές που θέλουμε να πουλήσουμε σε τιμή των μετοχών που είχαν μείνει, αφαιρούμε τις μετοχές που είχαμε από πριν με το ποσό που πουλήσαμε. Αν αυτές που έμειναν δεν είναι περισσότερες από αυτές που θέλουμε να πουλήσουμε τότε πουλάμε όσες είχαν μείνει (αν είναι περισσότερες από 0) και μετά αφαιρούμε από το ποσό που θέλουμε να πουλήσουμε το ποσό που είχαμε πριν και συνεχίζουμε. Μετά παίρνουμε το επόμενο Pair μετοχών που είναι να πουληθούν και αν είναι περισσότερες από αυτές που έχουν μείνει να πουλήσουμε τότε τις πουλάμε και τις υπόλοιπες που μένουν τις προσθέτουμε στο savedAmount για να πουληθούν μετά αλλιώς πουλάμε όσες αγοράσαμε και τις αφαιρούμε από αυτές που θέλουμε να πουλήσουμε και συνεχίζουμε για να ελέγξουμε αν μπορούμε να πουλήσουμε τις υπόλοιπες.

Ερώτημα 4

Το testingfiles περιέχει τα αρχεία που χρειάζονται για να γίνει έλεγχος της σωστής λειτουργίας. Επίσης το src περιέχει δύο .jar αρχεία που χρησιμοποιούνται για να τρέξω JUnit Tests από cmd και τα τέστ περιέχονται στα αρχεία: NetBenefitTest.java, QueueTest.java, StackTest.java, TestTagMatching.java, τα οποία είναι test cases και τρέχουν στο cmd με τις εντολές

- javac -cp junit-4.13.1.jar;. QueueTest.java StackTest.java TestTagMatching.java NetBenefitTest.java
- java -cp junit-4.13.1.jar;hamcrest-core-1.3.jar;. org.junit.runner.JUnitCore QueueTest StackTest TestTagMatching NetBenefitTes

Τα αρχεία που χρειάζονται όμως δουλεύουν κανονικά και με το cmd χωρίς JUnit και για τις ασκήσεις 2,3 ορίζουμε το file path από το command line π.χ.

- javac TagMatching.java
- java TagMatching C:/path/to/file.txt