Μέρος Δ

a. Στο ερωτημα αυτο φτιαξιαμε δυο κλασεις που αντιπροσοπευουν αφηρημενους τυπους δεδομενων (ΑΤΔ) οι οποιες κανουν implement τις δυο διεπαφες (Ετσι ωστε να υλοποιησουμε τις μεθοδους που μας δοθηκαν). Φτιαξαμε μια κλαση Node<T> η οποια ειναι απαραιτητη για την δημιουργια μια μονης λιστας συνδεσης.(Ειναι ουσιαστικα ενα dataholder το οποιο εχει ενα δεικτη που διχνει σε αλλον Node και παει λεγοντας...).
Το θετικο ειναι οτι το documentation ηταν ηδη ετοιμο στα interface αυτα επομενως δεν χρειαστηκε να κανουμε κατι εξτρα.

StringStackImpl<T>

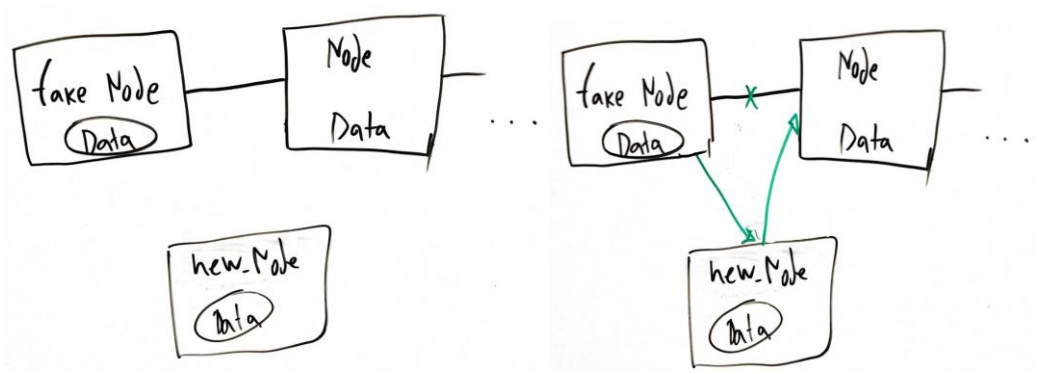
Σε αυτην την κλαση υλοποιησαμε το interface StringStack<T>. Λιγα λογια για την υλοποιηση των μεθοδων:

- **public boolean isEmpty();**

  Στη κλαση του ΑΤΔ που εχουμε δημιουργησει εχουμε ενα field τυπου integer που αναπαριστα το μεγεθος τις μονης λιστας συνδεσης. Ετσι λοιπον, αυτη η μεθοδος επιστρεφει true μονο οταν το field αυτο ισουτε με το 0.

- **public void push(T item);**

  Φτιαχνει ενα Node<T> το κανει populate με τα στοιχεια που δινονται (παραμετρος Item) και στην συνεχεια παει και το «χωνει» αναμεσα στον ψευδοκομβο και στον 1^{ο} πραγματικο κομβο. Αν ειστε visual learners ριξτε μια ματια στην απο κατω εικονα :



- **public T pop() throws NoSuchElementException;**

  Πεταει NoSuchElementException οταν η λιστα μονης συνδεση δεν εχει πραγματικα στοιχεια(!!προσοχη μπορει να εχει ψευδοκομβους).

Αυτο προγραμματιστικα το κανουμε με το να αλλαζουμε τον δεικτη του ψευδοκομβου να δειχνει στο επομενο απο τον κομβο που προκειτε να διαγραψουμε. Τα περιεχομενα data αυτου του κομβου τα επιστρεφει η συναρτηση.

- **public T peek() throws NoSuchElementException;**

Ακριβως το ιδιο σπεκτικο με την pop μονο που δεν διαγραφουμε τον 1^{ο} κομβο.

- **public void printStack(PrintStream stream);**

Ξεκιναμε απο τον 1^{ο} κομβο, τυπωνουμε τα δεδομενα του στο PrintStram και στην συνεχεια με την βοηθεια του 1^{ο} κομβου βρισκουμε τον 2^{ο} κομβο και κανουμε την ιδια επαναληπτικη διαδικασια.

- **public int size();**

Αυτη η μεθοδος υπολληποιηθηκε απλα να επιστρεφει το field size.

IntQueueImpl<T>

Σε αυτην την κλαση υπολοιησαμε το Interface IntQueue <T>. Λιγα λογια για την υλοποιηση των μεθοδων:

- **public boolean isEmpty();**

Το ιδιο σκεπτικο με την υλοποιηση του isEmpty() στην *StringStackImpl*<T>

- **public void put(T item);**
  Φτιαχνει ενα Node<T> με δεδομενο item και στην συνεχεια παει στο τελευταιο στοιχειο (head) της μονης λιστας συνδεσης και του αλλαζει το δεικτη να δειχνει στο νεο Node<T> που κατασκευαστηκε.

- **public T get() throws NoSuchElementException;**

  Αν η λιστα μονης συνδεσης δεν εχει πραγματικους κομβους τοτε σκαει NoSuchElementException**.**(Δηλαδη οταν head.next == null) .

  Αλλιως αν υπαρχουν στοιχεια γυρναει το πιο κοντικο στο head κομβο και στην συνεχεια το διαγραφει με την διαδικασια που ξερουμε.

- **public T peek() throws NoSuchElementException;**

Ακριβως ιδια λογικη οπως της get() μονο σε αυτην την μεθοδο δεν διαγραφεται το node<T> του οποιου κρυφοκοιταμε τα δεδομενα.

- **public void printQueue(PrintStream stream);**

  Τυπωνει στο PrintStream ολα τα στοιχεια των κομβων. Ξεκιναμε απο το ψευδοκομβο head,παμε στον επομενο Node<T> και τυπωνουμε τα δεδομενα του, στην συνεχεια με την βοηθεια αυτου βρισκουμε τον επομενο και συνεχιζουμε..

- **public int size();**
  Το ιδιο σκεπτικο με την size() του StringStackImpl<T>.

b. Για την επιλυση του ερωτηματος αυτου χρησιμοποιησαμε τον ΑΤΔ στοιβας που φτιαξαμε στο ερωτημα Α. Η χρηση στοιβας σε αυτο το ερωτημα εφαρμοζεται γαντι αφοτου μιλαμε για Parsing.

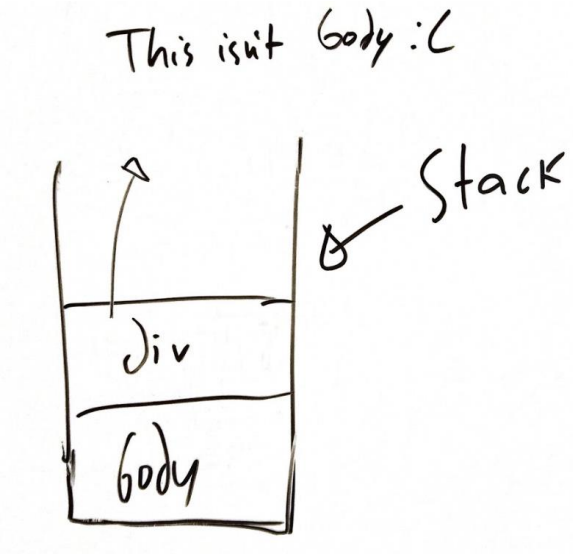
   Πως την χρησιμοποιησαμε; Πολυ απλα, γνωριζουμε οτι στο αρχειο html τα tags εχουν κατι κοινο ξεκινουν και κλεινουν με το ιδιο τροπο(<tag> </tag>)

   Εκμεταλευοντας αυτην την παρατηρηση διαβασαμε το html γραμμα ανα γραμμα και στην συνεχεια εαν επροκειτο για tag ανοιγματος τοτε το καναμε push στην στοιβα μας. Αντιθετα, εαν επροκειτο για tag κλεισιματος τοτε καναμε pop το στοιχειο απο την λιστα ελεγχοντας βεβαια οτι τα string ειναι ιδια.

Για παραδειγμα εστω οτι ειχαμε αυτο το html αρχειο:

```
11
12  <div>
13      <div>
14          <div>
15              <ul>
16                  <li></li>
17                  <li></li>
18                  <li></li>
19                  <li></li>
20              </ul>
21          </div>
22      </div>
23  </div>
24
25  <div> <!-- oops buddy you forgot to close that div there! -->
26
27  </body>
28
```

Τοτε αν μπορουσαμε να δουμε πως ειναι η στοιβα στην γραμμη 25 θα ηταν καπως ετσι:

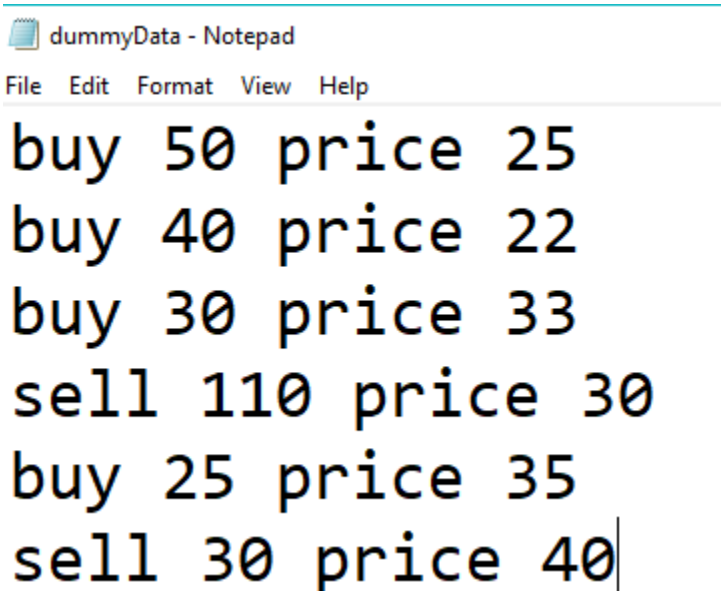


Ομως στο τελος ζηταμε να κανει pop απο το stack μονο που ο ελεγχος αν ιδιων string δεν ισχυει (div!= body) εδω και επομενως το προγραμμα επιστρεφει οτι οι ετικετες δεν ειναι ταιριασμενες.

c. Για την επιλυση του ερωτηματος αυτου χρησιμοποιησαμε τον ΑΤΔ ουρας που φτιαξαμε στο ερωτημα Α. Το προβλημα μπορουσε να λυθει και με αλλους τροπους που δεν περιειχαν ουρες ομως εμεις εχουμε να πουμε οτι με την χρηση ουρων το προβλημα ηταν αρκετο πιο ευκολο και καθαρο.

Για να γινει κατανοητο πως ακριβως χρησιμοποιησαμε τις ουρες θα χρησιμοποιησουμε ενα παραδειγμα:

Εστω οτι μας παρουμε αυτο το txt αρχειο με μετοχιακες πληροφοριες

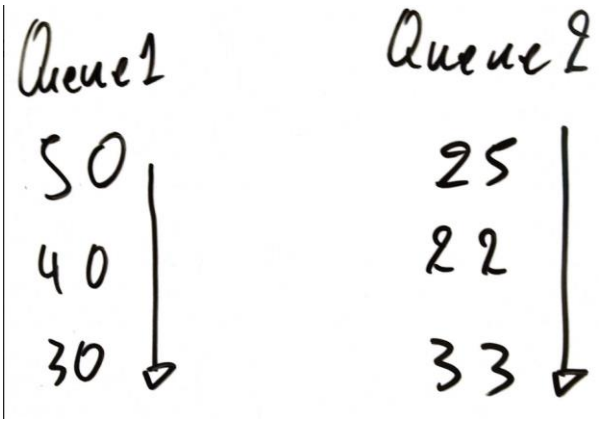


Για την αποθηκευση των δεδομενων χρησιμοποιουμε δυο ουρες οι αποιες εχουν στενη σχεση μεταξυ τους . Η μια ουρα αποθηκευει την ποσοτητα μετοχων ενω η αλλη την τιμη τους.

Ξεκιναμε να διαβασουμε το κειμενο γραμμη γραμμη και διακρινουμε τις εξεις περιπτωσεις:

Οταν ξεκιναει με buy πρεπει αν αποθηκευουμε δεδομενα στις ουρες μας ενω οταν ξεκιναει με sell δεν προσθετουμε δεδομενα αλλα αντιθετα μπορει και αν διαγραψουμε(οταν η ποσοτητα μετοχων γινει 0)

Οταν διαβασουμε τις πρωτες 3 γραμμες οι δυο ουρες μας θα εχουν αυτην την μορφη



Εφοσον εγινε κατανοητο πως αποθηκευουμε τα δεδομενα ας δουμε πως τα χρησιμοπουμε και ολας. Εστω τωρα οτι διαβασουμε την εντολη στην γραμμη 4 (πουλα 110 σε τιμη 30) τοτε αυτο που θα γινει ειναι οτι:

Θα παρουμε και απο τις δυο ουρες τις τα παλαιοτερα στοιχεια . Αν οι μετοχες παλαιωτερων στοιχειων δεν φτανουν τοτε τις κανουμε pop και προχοραμε στις αλλες. Προσοχη αυτο που αξιζει να σημειωθει ειναι οτι οταν οι δυο ουρες δεν εχουν στοιχεια πετανε exception το οποιο πιανουμε στο κυριο προγραμμα μας! Επισης οταν τελειωσουν οι μετοχες μια γραμμα στην στοιβα (δηλαδη οταν η ποσοτητα ειναι 0) δεν ξεχναμε αν διαγραψουμε και το τελευταιο στοιχιο της δευτερης ουρας που αναπαριστα την τιμη. Ετσι λοιπον εχουμε χτισει ενα robust προγραμμα με μια απλη βαση δεδομενο and that's all!!