



ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ - Η.Μ.Μ.Υ
ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΑΡΧΕΙΩΝ – ΠΛΗ202

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 1

ΑΤΟΜΙΚΗ ΑΝΑΦΟΡΑ ΥΛΟΠΟΙΗΣΗΣ

Παντελής Κωνσταντίνος
2015030070

Σκοπός Εργασίας:

Εξοικείωση με πασιφηλείς πρακτικές συγγραφής κώδικα σε Java(ή άλλη γλώσσα O.O.P) , A.D.T Λίστες, Τεχνικές Αναζήτησης και Αρχεία Δεικτοδότησης.

Περί Υλοποίησης:

Ο κώδικας για την παρούσα άσκηση γράφθηκε στην γλώσσα Java (***jdk 13.0.1***) στο IDE Eclipse, ενώ η ανάλυση και υλοποίηση αυτής κατατμίζεται σε 3 διακριτά υποτμήματα:

- **Μέρος Α :** *Διπλά Διασυνδεδεμένη Λίστα, Λειτουργίες Αρχείων.*
- **Μέρος Β:** *Αρχείο Δεικτοδότησης και Αναζητήσεις (Σειριακή, Δυαδική).*
- **Μέρος Γ:** *Πειράματα - Σύγκριση τεχνικών αναζήτησης.*

Κατακερματισμός Ζητούμενων - Υλοποίησης:

Μέρος Α:

- Δημιουργία line based text editor μέσω διπλά διασυνδεδεμένης λίστας (δυναμικώς υλοποιημένη).
- Υλοποίηση βασικών λειτουργιών σε αρχεία (ανάγνωση, εγγραφή).
- Διαχωρισμός πληροφορίας κειμένου (έγινε με χρήση ***Regular Expressions***).

Μέρος Β:

- Δημιουργία αρχείου δεικτοδότησης πληροφοριών (μορφή <Λέξη, Γραμμή>).
- Υλοποίηση λειτουργίας Σειριακής και Δυαδικής αναζήτησης (στο αρχείο δεικτοδότησης)

Μέρος Γ:

- Δοκιμαστικές Αναζητήσεις υπαρκτών και μη κλειδιών.
- Ανάλυση και επεξήγηση αποτελεσμάτων.

Οργάνωση Κώδικα και Documentation:

Ο πηγαίος κώδικας βρίσκεται οργανωμένος σε 5 πακέτα (packages) τα περιεχόμενα των οποίων περιγράφονται αναλυτικά στην παραχθείσα βιβλιογραφία (μορφή Javadoc) ενώ εντός του συμπιεσμένου (.zip) παραδοτέου υπάρχει και αρχείο με διάγραμμα κλάσεων (UML).

Package:	Summary:
<i>util</i>	Υλοποιήσεις χρήσιμων “εργαλείων” (ADT List, ADT Node, Word [tuple <String,Integer>], κ.α).
<i>Files</i>	Κλάσεις προσβάσεων σε Αρχεία (.txt αλλά και δεικτοδότησης).
<i>textEditor</i>	Κυρίως κλάση (Main.java) καθώς και κλάσεις που αφορούν τη διασύνδεση.
<i>searchOps</i>	Interface αναζήτησης και δύο κλάσεις υλοποίησης για αναζητήσεις (Σειριακή, Δυναμική).
<i>testing</i>	Κλάση για παραγωγή και αναζήτηση τυχαίων κλειδιών.

Υλοποίηση:

Μέρος Α:

“A line based text editor using a List and some functions”

Λίστα:

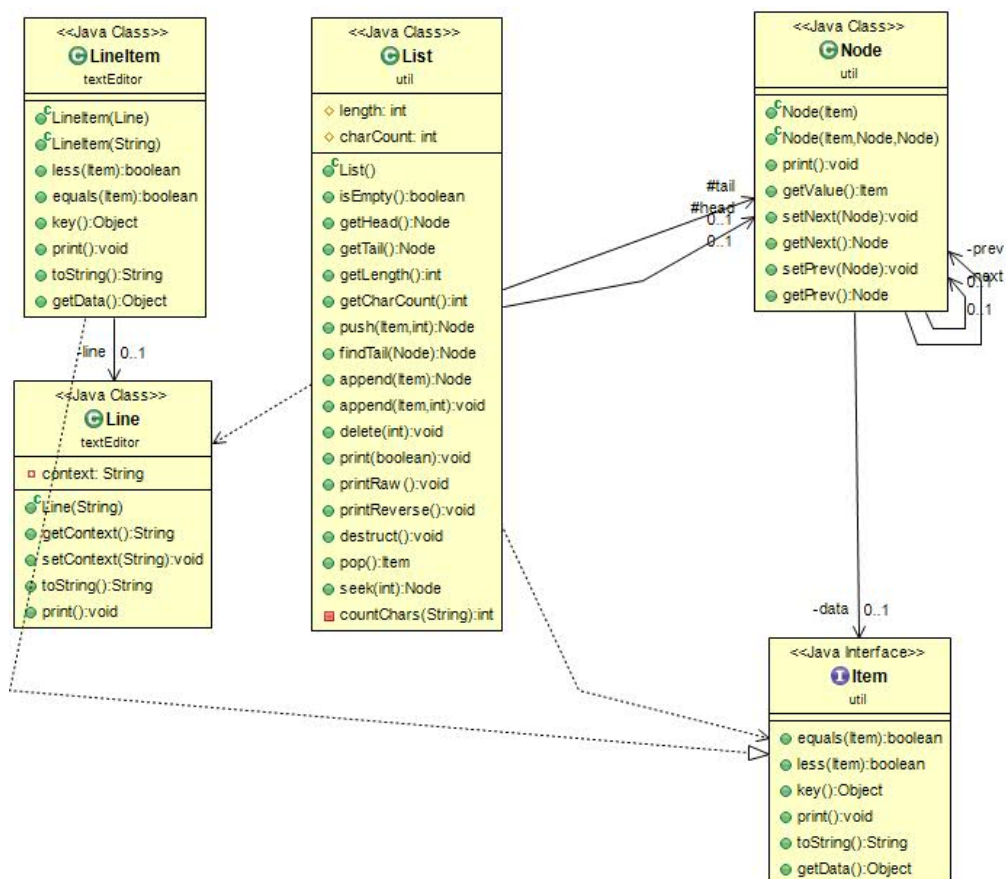
- *List.java, Node.java, Item.java, Line.java , LineItem.java*

Υλοποιήθηκε μια διπλά διασυνδεμένη λίστα με βάση και τις διαλέξεις του μαθήματος της οποίας κάθε κόμβος τυγχάνει δυναμικής δέσμευσης και περιλαμβάνει μία γραμμή κειμένου που θεωρείται ορθή ως εξής:

- Ορισμός Διεπαφής Item (Interface Item), κλάσεων Line, LineItem έτσι ώστε κάθε αντικείμενο κλάσης που υλοποιεί (implements) την Item να μπορεί να αποτελέσει στοιχείο ενός κόμβου (*).
- Ορισμός κλάσης κόμβου λίστας (Node.java) ώστε να περιλαμβάνει ένα οποιοδήποτε Item (εδώ μόνο αντικείμενα Line) καθώς και αναφορές σε επόμενο και προηγούμενο κόμβο.
- Ορισμός Λειτουργιών Λίστας (Εισαγωγή, Αναζήτηση, Διαγραφή, κ.α) σύμφωνα με τις προδιαγραφές της εκφώνησης.

(*): Η περιγραφείσα τεχνική επιτρέπει την ουσιαστική πληροφορία να επεκταθεί μελλοντικά και σε άλλες μορφές διαφορετικές της *Line/LineItem* (εφόσον φυσικά υλοποιεί τη διεπαφή *Item*). Αποτελεί δε πρακτική που διδάχθηκε στο “ΠΛΗ-102” ελαφρώς παραλλαγμένη.

Σηματικά:



Διάβασμα Αρχείου:

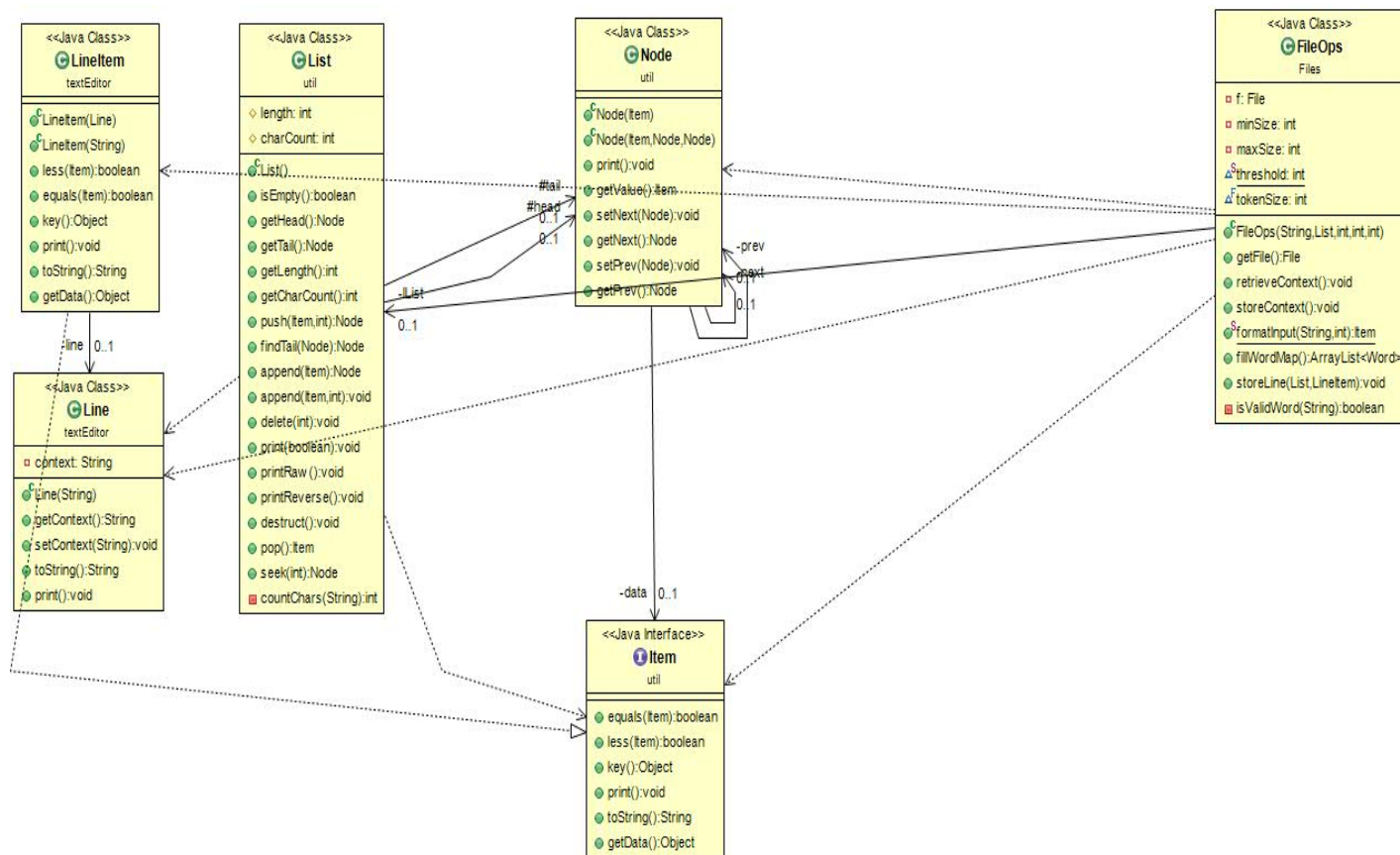
- *FileOps.java*

Προκειμένου να διαβασθούν αρχεία με ουσιαστική πληροφορία υλοποιήθηκε η κλάση FileOps στο πακέτο Files της οποίας η κυρίως λειτουργικότητα ακολουθεί:

- Διάβασμα γραμμή – γραμμή αρχείου .txt που του οποίου το όνομα δίδεται ως String (αν δε δοθεί θα παραχθεί αρχείο με τυχαίο όνομα αρχικά κενό).
- Έλεγχος γραμμής και αποθήκευση στη λίστα με τη σειρά εισαγωγής.
 - Κενές γραμμές αγνοούνται.
 - Γραμμές άνω των 80 χαρακτήρων περικόπτονται.
 - Μήνυμα σχετικά με την περικοπή ή την αγνόηση κενής γραμμής δε παρέχεται στο χρήστη.
- Αποθήκευση λίστας - γραμμών στο αρχικό αρχείο.
 - Πρόκειται ουσιαστικά για την γραμμή προς γραμμή εγγραφή στο αρχικό αρχείο του περιεχομένου της λίστας.
 - Τυχών αλλαγές στοιχείων συμπεριλαμβάνονται (προσθήκες, διαγραφές γραμμών) .
 - Το αρχικό περιεχόμενο του αρχείο “πανωγράφεται”.
- Δημιουργία ταξινομημένου ArrayList με ζεύγη <Λέξη, Γραμμή> με βάση τις γραμμές κειμένου στοιχεία της λίστας (*).
 - Απαραίτητο για το Β’ Μέρος.
 - Χωρισμός γραμμών με χρήση Regular Expression σε λέξεις.
 - Έλεγχος λέξης αναφορικά με τα επιτρεπτά όρια μήκους.
 - Λέξεις με μήκος μεγαλύτερο των 20 και μικρότερο των 5 χαρακτήρων αγνοούνται ως μη έγκυρες.
 - Αποθήκευση έγκυρων λέξεων σε ζεύγη <Λέξη, Γραμμή> ως αντικείμενα κλάσης (Word.java) σε ArrayList το οποίο ταξινομείται.

(*): Η υλοποίηση του θα περιγραφεί αναλυτικότερα στο Β’ Μέρος.

Σημιατικά:



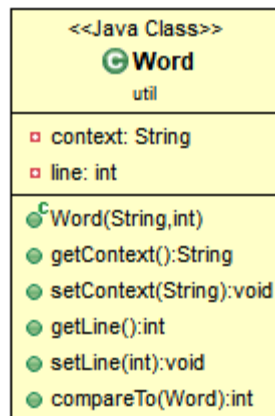
Σύνδεση και τελική λειτουργικότητα:

Υλοποιήθηκε η κλάση `Main.java` η οποία αναλαμβάνει τον έλεγχο εισόδου και την σύνδεση μεταξύ κλάσεων για την επίτευξη της επιθυμητής λειτουργικότητας του Μέρους Α, καθώς και η αρχική λειτουργικότητα του Μέρους Β' η οποία παρατίθεται ακολούθως δια το ορθότερο της περιγραφής.

Η κλάση Word (Word.java) :

Πρόκειται για την προγραμματιστική απεικόνιση – μοντελοποίηση ζευγαριών <Λέξη, Γραμμή> η οποία υλοποιεί τη διεπαφή (Interface) Comparable για αντικείμενα της κλάσης της.

Συγκεκριμένα η δήλωσή της έχει ως εξής:



Σημαντικό στοιχείο ο τρόπος υλοποίησης της μεθόδου `compareTo`, καθότι μέσω αυτής καθίσταται δυνατή η διάταξη (άρα και η αλφαριθμητική ταξινόμηση) αντικειμένων της `Word` (οργανωμένων σε δομή `ArrayList` ή πίνακα) με βάση κάποιο κριτήριο και μία μεταβλητή μέλος (επιλέχθηκε το `context`).

Άξιο αναφοράς ότι η διάταξη με βάση το `String context` είναι *case – sensitive* όπως ζητείται ενώ για ταυτοτικά `Strings` η διάταξη συμβαίνει με κριτήριο τον αριθμό γραμμής (βλ. `Javadoc`, πηγ. Κώδικα).

Μέρος Β’:

“Αρχείο δεικτοδότησης και αναζητήσεις”

Δημιουργία Αρχείου δεικτοδότησης:

- *FileOps.java*
- *FilePageAccess.java*

Δεδομένου ταξινομημένου ArrayList με αντικείμενα Word ακολουθείται η εξής διαδικασία για την κατασκευή του αρχείου δεικτοδότησης (RandomAccessFile):

- Κάθε ζευγάρι <String word, Integer Line> μετατρέπεται σε bytes με την τεχνική που προτάθηκε στην εκφώνηση και γράφεται στο αρχείο ως εξής:
 - Δεσμεύεται χώρος pageSize (εδώ 128 bytes) σε έναν buffer στον οποίο και γράφονται οι πληροφορίες ως bytes έως ότου ο buffer γεμίσει.
 - Κατόπιν γράφουμε τον buffer στο αρχείο δεικτοδότησης.
 - Ένα “πακέτο” (<String, Integer> αντικείμενο της κλάσης Word) καταλαμβάνει μέγιστο χώρο ίσο με:

$$\text{Μέγεθος Λέξης(μέγιστο)} + \text{Μέγεθος Γραμμής} = 20 + 4 = 24 \text{ bytes.}$$

- Σε περίπτωση που το “κλειδί” - String καταλαμβάνει λιγότερο πραγματικό μέγεθος γεμίζουμε τον υπολοιπόμενο χώρο με spaces (ASCII 32 / byte_{HEX} = 20).

Παράδειγμα:

Έστω **Word = new Word(“Each”, 3)** το αντίστοιχο κωδικοποιείται ως εξής:

20 bytes για τη λέξη Each						4 bytes για τη θέση 3				
E dec 69	a dec 97	c dec 99	h dec 104	Space dec 32	Space dec 32	dec 0	dec 0	dec 0	dec 3
01000101	01100001	01100011	01101000	00100000		00100000	00000000	00000000	00000000	00000011

- Σε buffer **128 bytes** μπορούν να αποθηκευθούν το πολύ **$128 \div 24 = 5$ ζεύγη**. Για **pageSize** γενικά αποθηκεύουμε **pageSize \div packetSize** κλειδιά το πολύ.

Οπότε:

Έχουμε $(Integer) \frac{pageSize}{packetSize}$ κλειδιά /σελίδα ή $\frac{128}{24} = 5$ κλειδιά/σελίδα δεδομένου ότι τα μεγέθη έχουν ως εξής:


Μεταβλητή	Μέγεθος(bytes)
Λέξη	20 (ίσως space padded)
Γραμμή(Integer)	4
Word Object	20+4 = 24
Σελίδα	128

Διάβασμα Αρχείου δεικτοδότησης:

- **FilePageAccess.java**

Διαδικασία όμοια με αυτή της εγγραφής αναφορικά με την τεχνική αποκωδικοποίησης όσο και με το μέγεθος του buffer στον οποίο διαβάζονται δεδομένα σελίδα – σελίδα (5- αδες εδώ). Επιπλέον καταγράφονται οι προσβάσεις στο δίσκο για το διάβασμα έκαστης σελίδας.

Σημειώνεται οτι η μείωση κατα το δυνατό των αυτών προσβάσεων είναι αυτή που πρακτικά θα καθορίσει την απόδοση της μεθόδου αναζήτησεως σε επόμενο βήμα.

<<Java Class>>	
 FilePageAccess	
Files	
<div>□ pageSize: int</div> <div>□ tokenSize: int</div> <div>□ dataPages: int</div> <div>□ keySize: int</div> <div>▲ dF: RandomAccessFile</div>	
<div>● FilePageAccess(int,int,String)</div> <div>● getDataPages():int</div> <div>● getPageSize():int</div> <div>● fillDictionary(ArrayList<Word>):void</div> <div>● readDictionary():ArrayList<Word></div> <div>● readPage(int):ArrayList<Word></div> <div>■ paddBytes(byte[]):byte[]</div> <div>● close():void</div> <div>● printFile():void</div>	

Τεχνικές Αναζήτησης σε Αρχείο:

- *package searchOps*

Σκοπός η υλοποίηση δύο πασιφιλών τεχνικών αναζήτησης σε Δομές Δεδομένων (συγκεκριμένα σε αρχείο) , της Σειριακής και της Δυναδικής. Εν γένει όπως προαναφέρθηκε ζητούμενο εδώ αποτέλεσε, πλέον της προγραμματιστικής υλοποίησης, η εξαγωγή συμπερασμάτων σχετικά με την αποδοτικότερη τεχνική (σε όρους προσβάσεων στη μνήμη – δίσκο).

Συνεπώς θα πρέπει να ληφθούν υπόψιν τα κάτωθι:

- Τα κλειδιά που γράφονται στο αρχείο είναι ταξινομημένα (χάριν συντομίας τα ζεύγη – Word Objects θα καλούνται πλέον “κλειδιά”).
- Τα κλειδιά δεν είναι υποχρεωτικά μοναδικά. Μια λέξη μπορεί να έχει πολλαπλές αναφορές, δηλαδή να εντοπίζεται σε αρκετές γραμμές του αρχικού αρχείου. Συνεπώς στην ακραία περίπτωση θα μπορούσε να καταλαμβάνει πάνω απο μία σελίδες του αρχείου δεικτοδότησης (αυτό συνεπάγεται αύξηση προσβάσεων στο worst case scenario).

Υλοποιήθηκαν λοιπόν με βάση τα παραπάνω:

- Διεπαφή – Interface fileSearch, η οποία καθορίζει τη δομή κάθε κλάσης αναζήτησης.
- Κλάσεις SerialFileSearch, BinaryFileSearch που υλοποιούν την Σειριακή και Δυναδική αναζήτηση.

Σχετικά με τις κλάσεις SerialFileSearch, BinaryFileSearch:

- Διαβάζουν σελίδα προς σελίδα το αρχείο δεικτοδότησης.
- Καταγράφουν προσβάσεις, αναζητούν κλειδί, εκτυπώνουν ευρέσεις.

Είναι λοιπόν όμοιες αναφορικά με τη δομή τους, (λογικό) δεδομένου οτι υλοποιούν την αρχική διεπαφή, διαφέρουν ωστόσο στον αλγόριθμο αναζήτησης με βάση τον οποίο προσπελαίνουν σελίδες στο αρχείο (συνάρτηση **search(String key)**) ο οποίος περιγράφεται παρακάτω.

Σειριακή Αναζήτηση:

Φορτώνει το αρχείο σελίδα – προς – σελίδα στον buffer ξεκινώντας απο την πρώτη σελίδα. Ελέγχει κάθε σελίδα στο buffer για την ύπαρξη του κλειδιού. Εύρεση του κλειδιού στην τελευταία θέση του buffer, προκαλεί την φόρτωση και της επόμενης σελίδας προκειμένου να μην απολεσθεί τυχόν επανεμφάνισή του (corner case – worst case).

Δυναδική Αναζήτηση:

Φορτώνει στον buffer την μεσαία σελίδα του αρχείου. Εύρεση του κλειδιού στην πρώτη ή την τελευταία θέση του buffer προκαλεί την φόρτωση της προηγούμενης ή επόμενης σελίδας ανάλοφα προκειμένου να μην απολεσθεί τυχόν επανεμφάνιση του κλειδιού στο αρχείο. Μη εύρεση του κλειδιού στην τρέχουσα σελίδα προκαλεί φόρτωση της μεσαίας σελίδας του αριστερού ή δεξιού μισού του αρχείου.

Η φόρτωση του αριστερού ή δεξιού μέρους εξαρτάται απο την λεξικογραφική σύγκριση του κλειδιού με το πρώτο και το τελευταίο στοιχείο της σελίδας στην οποία αυτό δε βρέθηκε.

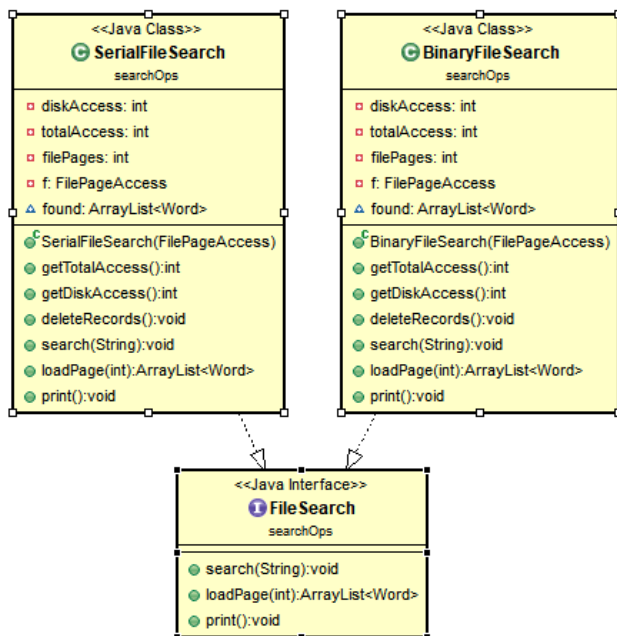
Δηλαδή:

Key < first_key_of_page → Προχωράμε αριστερό μισό.

Key = first_key_of_page → Προχωράμε μια θέση αριστερά (key might be duplicate).

Key > last_key_of_page → Προχωράμε δεξιό μισό.

Key = last_key_of_page → Προχωράμε μια θέση δεξιότερα (key might be duplicate).



Πείραμα:

1) Αναζήτηση για τα κλειδιά “Rectors”, “laboratories”, “Technical”, “Venetian”

Σειριακή:

Αρχείο(testfile-x.txt)	Προσβάσεις	Προσβάσεις	Προσβάσεις	Προσβάσεις
	<u>Rectors</u>	<u>laboratories</u>	<u>“Technical”</u>	<u>“Venetian”</u>
1	38	92	46	53
x2	75	183	91	106
x5	188	457	227	264
x10	375	915	453	527
x1000	37401	91401	45201	52601

Διαδική:

Αρχείο(testfile-x.txt)	Προσβάσεις	Προσβάσεις	Προσβάσεις	Προσβάσεις
	<u>Rectors</u>	<u>laboratories</u>	<u>“Technical”</u>	<u>“Venetian”</u>
1	7	6	7	5
x2	7	8	8	8
x5	9	9	10	9
x10	11	11	10	11
x1000	17	17	17	17

2) Ανεπιτυχής Αναζήτηση 30 τυχαίων κλειδιών μήκους 5 έως 20 χαρακτήρων.

- *TestCase_1.java*

Αρχείο(testfile-x.txt)	Σειριακή – Μ.Ο	Δυναδική – Μ.Ο
1	130	6
x2	258	7
x5	645	9
x10	1290	10
x1000	129000	16

Μέρος Γ: “Σχολιασμός Αποτελεσμάτων - Συμπεράσματα”

Σκοπός του παρόντος εδαφίου αποτελεί η σύγκριση των τεχνικών Σειριακής και Δυναδικής αναζήτησης στην εφαρμογή τους σε **ταξινομημένα αρχεία – δομές**.

Βασιζόμενοι στους πίνακες σύγκρισης προσβάσεων τόσο για επιτυχείς αναζητήσεις υπαρκτών κλειδιών όσο και για ανεπιτυχείς τυχαίων κλειδιών μπορεί εύκολα να εξαχθεί το γεγονός ότι, η Δυναδική Αναζήτηση είναι τάξεις μεγέθους αποδοτικότερη σε όρους προσβάσεων (άρα και ταχύτητας και υπολογιστικού φόρτου) στο αρχείο δεικτοδότησης.

Το παραπάνω γεγονός έγκειται στη φύση των αναζητήσεων καθότι **η Σειριακή – Εξαντλητική** αναζήτηση **δεν εκμεταλλεύεται το στοιχείο της ταξινόμησης του αρχείου** (βλ. Αρχικές παραδοχές) προσπελώντας όλα τα στοιχεία στη χειρότερη περίπτωση (μη εύρεση κλειδιού).

Τουναντίον η τεχνική της Δυναδικής Αναζητήσεως εκμεταλλεύεται το αυτό γεγονός χωρίζοντας τη Δομή (Αρχείο) σε υπο – Δομές (αριστερό, δεξιό μισό κτλ.) στις οποίες **κινείται κατευθυνόμενα ελαχιστοποιώντας τις προσβάσεις (αγνοεί την μισή δομή απο την πρώτη κίολας σύγκριση)** με αποτέλεσμα τη μεγιστοποίηση της απόδοσής της, ως τεχνική, για ταξινομημένα αρχεία – δομές.

Σε επίπεδο πολυπλοκότητας δε βλέπουμε την Σειριακή να ακολουθεί γραμμική αύξηση προσβάσεων – κλειδιών κάτι που συνάδει με την θεωρητική $O(n)$, ενώ αναφορικά με τη Δυναδική η θεωρητική πολυπλοκότητα $O(\log(n))$ δείχνει επίσης να επιβεβαιώνεται.

Παραπομπές – Πηγές:

1. <https://www.geeksforgeeks.org/generate-random-string-of-given-size-in-java/>
2. Φροντιστήριο Μαθήματος.
3. Σελίδα Μαθήματος Courses.
4. Άρθρα στο [geegksforgeeks.org](https://www.geeksforgeeks.org)