

**Υλοποίηση Αλγορίθμου για τον Υπολογισμό του Ελάχιστου Περικλείοντος Κύκλου και του Διαγράμματος Voronoi**

**Πρίφτη Ιωάννης**

**Διπλωματική Εργασία**

Επιβλέπων: Παληός Λεωνίδας

Ιωάννινα, Φεβρουάριος, 2023

**Τμήμα Μηχ. Η/Υ & Πληροφορικής**

**Πανεπιστήμιο Ιωαννίνων**

**Department of Computer Science & Engineering**

**University of Ioannina**

Ευχαριστίες

Θα ήθελα όλους τους ανθρώπου που συνέβαλαν και βοήθησαν στο να γίνω ο άνθρωπος που είμαι σήμερα.

20/02/2023

Πρίφτη Ιωάννης

Περίληψη

Η εργασία αυτή αφορά την υλοποίηση και οπτικοποίηση του αλγορίθμου για τον υπολογισμό του Ελάχιστου Περικλείοντος Κύκλου καθώς επίσης και των αντίστοιχων διαγραμμάτων Voronoi για τους κοντινότερους και μακρύτερους γείτονες διατηρώντας πάντα τις πολυπλοκότητες των αλγορίθμων σε χρονικό πλαίσιο O(nlogn). Για την ανάπτυξη χρησιμοποιήθηκε η γλώσσα προγραμματισμού java καθώς επίσης και javaFX. Τέλος, αποτελεί μια πλήρη εφαρμογή η οποία μπορεί να γίνει εγκατάσταση.

**Λέξεις Κλειδιά:** Ελάχιστος περικλείοντας κύκλος, Διάγραμμα Voronoi, Πολυπλοκότητα, Εφαρμογή

**Πίνακας Περιεχομένων**

[Κεφάλαιο 1. Εισαγωγή 8](#_Toc127716623)

[1.1 Βασικοί Ορισμοί 8](#_Toc127716624)

[1.2 Αντικείμενο της Διπλωματικής Εργασίας 8](#_Toc127716625)

[1.3 Σχετικά Ερευνητικά Αποτελέσματα 9](#_Toc127716626)

[1.4 Δομή της Διπλωματικής Εργασίας 9](#_Toc127716627)

[Κεφάλαιο 2. Υπολογισμός Ελάχιστου Περικλείοντος Κύκλου και του Διαγράμματος Voronoi 10](#_Toc127716628)

[2.1 Υπολογισμός Κυρτού Περιβλήματος 10](#_Toc127716629)

[2.1.1 Θεωρητικό Υπόβαθρο 10](#_Toc127716630)

[2.1.2 Περιγραφή του Αλγορίθμου 10](#_Toc127716631)

[2.1.3 Παράδειγμα Εφαρμογής του Αλγορίθμου 11](#_Toc127716632)

[2.2 Υπολογισμός Ελάχιστου Περικλείοντος Κύκλου 13](#_Toc127716633)

[2.2.1 Θεωρητικό Υπόβαθρο 14](#_Toc127716634)

[2.2.2 Περιγραφή του Αλγορίθμου 14](#_Toc127716635)

[2.2.3 Παράδειγμα Εφαρμογής του Αλγορίθμου 14](#_Toc127716636)

[2.3 Υπολογισμός Διαγράμματος Voronoi 14](#_Toc127716637)

[2.3.1 Περιγραφή του Αλγορίθμου 14](#_Toc127716638)

[2.3.2 Παράδειγμα Εφαρμογής του Αλγορίθμου 14](#_Toc127716639)

[2.4 14](#_Toc127716640)

[Κεφάλαιο 3. Η Υλοποίηση 16](#_Toc127716641)

[3.1 Είσοδος-Έξοδος 16](#_Toc127716642)

[3.2 Δομές Δεδομένων 16](#_Toc127716643)

[3.3 Κλάσεις/Πακέτα 17](#_Toc127716644)

[3.3.1 MainGUI/View 17](#_Toc127716645)

[3.3.2 SmallestEnclosingCircleGUI/View 23](#_Toc127716646)

[3.3.3 VoronoiGraphGUI/View 29](#_Toc127716647)

[3.3.4 AlgorithmsHandler/Control 35](#_Toc127716648)

[3.3.5 ChangeLanguageButtonHandler/Control 37](#_Toc127716649)

[3.3.6 CustomMouseListener/Control 39](#_Toc127716650)

[3.3.7 ExitButtonHandler/Control 41](#_Toc127716651)

[3.3.8 GraphGUIHandler/Control 41](#_Toc127716652)

[3.3.9 InfoButtonHandler/Control 43](#_Toc127716653)

[3.3.10 InfoContentHandler/Control 47](#_Toc127716654)

[3.3.11 LoadFileButtonHandler/Control 48](#_Toc127716655)

[3.3.12 PaperTitleHandler/Control 50](#_Toc127716656)

[3.3.13 PutPointsButtonHandler/Control 51](#_Toc127716657)

[3.3.14 DataBase/Model 52](#_Toc127716658)

[3.3.15 GrahamScan/Model 68](#_Toc127716659)

[3.3.16 NeighborVoronoiDiagram/Model 71](#_Toc127716660)

[3.3.17 SmallestEnclosingCircle/Model 73](#_Toc127716661)

[3.3.18 TextHandler/Model 74](#_Toc127716662)

[3.3.19 CSVLoader/Model\_Loaders 81](#_Toc127716663)

[3.3.20 ExcelLoader/Model\_Loader 83](#_Toc127716664)

[3.3.21 FileLoader/Model\_Loader 84](#_Toc127716665)

[3.3.22 FileLoaderFactory/Model\_Loaders 85](#_Toc127716666)

[3.3.23 TXTLoader/Model\_Loader 87](#_Toc127716667)

[3.4 Παραδείγματα Εκτέλεσης του Προγράμματος 88](#_Toc127716668)

[Κεφάλαιο 4. Επίλογος 90](#_Toc127716669)

[Κεφάλαιο 5. Βιβλιογραφία 91](#_Toc127716670)

# Εισαγωγή

## Βασικοί Ορισμοί

Σε αυτήν την ενότητα θα αναλύσουμε τους βασικούς ορισμούς που αφορούν τον το Κυρτό Πολύγωνο, τον Ελάχιστο Περικλείοντα Κύκλο καθώς και των διαγραμμάτων Voronoi για τον κοντινότερο και μακρύτερο γείτονα.

Το Κυρτό Πολύγωνο είναι το μικρότερο κυρτό σύνολο που περιέχει όλα τα σημεία στον ευκλείδειο χώρο, πιο συγκεκριμένα είναι εκείνο το πολύγωνο που έχει ως κορυφές του τα πιο εξωτερικά σημεία του ευκλείδειου χώρου με την ιδιότητα να καλύπτει όλα τα υπόλοιπα και ταυτόχρονα να είναι το μικρότερο δυνατό.

Ο Ελάχιστος περικλείοντας Κύκλος είναι ο Κύκλος με την μικρότερη ακτίνα που περιέχει όλα τα σημεία στον ευκλείδειο χώρο και κάποια εξ ’αυτών(2-3 συνήθως) βρίσκονται πάνω στον κύκλο.

Το Διάγραμμα Voronoi του κοντινότερου γείτονα(ή 1ης τάξης) είναι ένα διάγραμμα το οποίο χωρίζει τον χώρο σε περιοχές(πολύγωνα ή κελιά Voronoi). Σε κάθε σημείο του ευκλείδειου επιπέδου αντιστοιχεί και μια περιοχή του διαγράμματος με την ιδιότητα ότι οποιοδήποτε σημείο βρεθεί εντός της περιοχής αυτής είναι πιο κοντά στο σημείο αυτό απ’ ότι σε όλα τα υπόλοιπα.

Το Διάγραμμα Voronoi του μακρύτερου γείτονα(ή ν-1τάξης) είναι ένα διάγραμμα που μας δείχνει το ακριβώς αντίθετο σε σχέση με του κοντινότερου γείτονα. Πιο συγκεκριμένα, χωρίζει τον ευκλείδειο χώρο σε περιοχές με τέτοιον τρόπο ώστε για οποιοδήποτε σημείο να μπορούμε να δούμε τον μακρύτερο γείτονά του.

## Αντικείμενο της Διπλωματικής Εργασίας

Σε αυτήν την διπλωματική εργασία είχαμε ως στόχο να μελετήσουμε και υλοποιήσουμε τον Αλγόριθμο για τον υπολογισμό του Ελάχιστου Περικλείοντος Κύκλου καθώς επίσης και των Διαγραμμάτων Voronoi του Κοντινότερου και Μακρύτερου γείτονα. Καταφέραμε να διατηρήσουμε την πολυπλοκότητα των Αλγορίθμων σε O(nlogn) χρησιμοποιώντας αποδοτικές δομές(όπως λεξικά, κοκκινόμαυρα 2-αδικά δένδρα αναζήτησης κτλ.) και δημιουργήσαμε μια εφαρμογή στην γλώσσα προγραμματισμού Java με εύχρηστο γραφικό περιβάλλον ικανή να μπορεί να την χρησιμοποιεί ο χρήστης χωρίς να απαιτείται γνώσεις προγραμματισμού.

## Σχετικά Ερευνητικά Αποτελέσματα

## Δομή της Διπλωματικής Εργασίας

# Υπολογισμός Ελάχιστου Περικλείοντος Κύκλου και του Διαγράμματος Voronoi

## Υπολογισμός Κυρτού Περιβλήματος

Σε αυτό το κεφάλαιο θα περιγράψουμε τον αλγόριθμο σάρωσης του Graham για τον υπολογισμό των σημείων που ορίζουν το κυρτό περίβλημα από ένα πεπερασμένο σύνολο σημείων σε O(n+nlogn) χρόνο καθώς επίσης και θα εξηγήσουμε μέσω ενός απλού παραδείγματος τον τρόπο με τον οποίο λειτουργεί.

### Θεωρητικό Υπόβαθρο

Ας υποθέσουμε λοιπόν ότι έχουμε ένα σύνολο από καρφιά καρφωμένα πάνω σε ένα ξύλο και θέλουμε να πάρουμε ένα λαστιχάκι ώστε αν το τεντώσουμε αρκετά να μπορούμε να καλύψουμε όλα τα καρφιά. Το όριο που θα σχηματιστεί από το λαστιχάκι θα έχει την ιδιότητα ότι κάθε εσωτερική γωνία θα είναι το πολύ 180ο και ότι όλα τα σημεία θα είναι είτε εντός του ορίου είναι ακριβώς πάνω στο όριο. Το όριο αυτό ονομάζεται κυρτό περίβλημα και ακριβώς αυτή είναι η λογική πίσω από τον αλγόριθμο του Graham.

### Περιγραφή του Αλγορίθμου

Το πρώτο βήμα του αλγορίθμου είναι η εύρεση του σημείου με την μικρότερη συντεταγμένη y και στην περίπτωση ισοπαλίας επιλέγεται εκείνο με την μικρότερη συντεταγμένη x σε O(n) χρόνο, έστω min.

Στη συνέχεια, ο αλγόριθμος ταξινομεί το σύνολο των σημείων σε αύξουσα σειρά με βάση την πολική γωνία(Math.atan2) που σχηματίζει το κάθε σημείο του συνόλου με το min ως προς τον άξονα x. Ο συγκριτής συγκρίνει πρώτα τις πολικές γωνίες των δύο σημείων και επιστρέφει -1, 0 ή 1 ανάλογα με το αν η γωνία του πρώτου σημείου είναι μικρότερη, ίση ή μεγαλύτερη από την γωνία του δεύτερου σημείου. Εάν οι πολικές γωνίες είναι ίσες, ο συγκριτής συγκρίνει τις αποστάσεις των δύο σημείων από το σημείο min και επιστρέφει το αποτέλεσμα τις σύγκρισης. Η ταξινόμηση γίνεται με τον Αλγόριθμο Merge Sort(Collections.short) σε O(nlogn) χρόνο.

Έπειτα, εκτελείται η σάρωση του Graham επί της λίστας(List<Point2D>) με τα ταξινομημένα στοιχεί που προέκυψαν στο προηγούμενο βήμα. Δημιουργεί μια νέα λίστα(List<Point2D>) που θα διατηρεί τα σημεία που ορίζουν το κυρτό περίβλημα στην οποία αρχικά προσθέτει το σημείο min. Για κάθε σημείο της ταξινομημένης λίστας ελέγχει αν το σημείο κάνει αριστερή στροφή με τα δύο προηγούμενα σημεία του κυρτού περιβλήματος από την αντίστοιχη λίστα τότε το προσθέτει στην αντίστοιχη λίστα με τα σημεία του κυρτού περιβλήματος. Διαφορετικά, αφαιρεί το προηγούμενο σημείο από το κυρτό περίβλημα μέχρι το τρέχον σημείο να κάνει αριστερή στροφή. Τέλος, ο αλγόριθμος προσθέτει ξανά το αρχικό σημείο στο κυρτό περίβλημα για να ολοκληρώσει τον βρόχο και επιστρέφει την λίστα. Ο έλεγχος της στροφής γίνεται μέσω της συνάρτησης crossProduct.

### Παράδειγμα Εφαρμογής του Αλγορίθμου

Ας υποθέσουμε ότι έχουμε 5 σημεία στον ευκλείδειο χώρο. Τα A, B, C και D. Εκ των οποίων επιλέγεται εκείνο με την μικρότερη συντεταγμένη y, δηλαδή το P.

Chart

Description automatically generated

**Σχήμα 1.1:** Επιλογή σημείου P

Αφού λοιπόν προστεθούν τα P και A στην λίστα με τα σημεία του κυρτού περιβλήματος, ο αλγόριθμος ελέγχει αν η ευθεία B-A σχηματίζει αριστερή στροφή με την ευθεία A-P.

Chart

Description automatically generated

**Σχήμα 1.2:** Έλεγχος στροφής B-A και A-P

Εφόσον σχηματίζουν αριστερή στροφή τότε προστίθεται το B στην λίστα με τα σημεία του κυρτού περιβλήματος και ο αλγόριθμος συνεχίζει στο επόμενο σημείο, το C. Ελέγχει αν η ευθεία C-B σχηματίζει αριστερή στροφή με την ευθεία B-A.

Chart, line chart
δσαψσδψσδ

Description automatically generated

**Σχήμα 1.3:** Έλεγχος στροφής C-B και B-A

Εφόσον σχηματίζουν αριστερή στροφή τότε προστίθενται το C στην λίστα με τα σημεία του κυρτού περιβλήματος και ο αλγόριθμος συνεχίζει στο επόμενο σημείο, το D. Ελέγχει αν η ευθεία D-C σχηματίζει αριστερή στροφή με την ευθεία C-B.



**Σχήμα 1.4:** Έλεγχος στροφής D-C και C-B

Εφόσον σχηματίζουν δεξιά στροφή τότε αφαιρείται το C από την λίστα με τα σημεία του κυρτού περιβλήματος και ο αλγόριθμος ελέγχει αν η ευθεία D-B σχηματίζει αριστερή στροφή με την ευθεία B-A.

Chart, line chart

Description automatically generated

**Σχήμα 1.5:** Έλεγχος στροφής D-B και B-A

Εφόσον σχηματίζουν αριστερή στροφή τότε προστίθενται το D στην λίστα με τα σημεία του κυρτού περιβλήματος και ο αλγόριθμος συνεχίζει στο επόμενο σημείο, το P. Ελέγχει αν η ευθεία P-D σχηματίζει αριστερή στροφή με την ευθεία D-B.

Chart, line chart, polygon

Description automatically generated

**Σχήμα 1.6:** Έλεγχος στροφής P-D και D-B

Εφόσον σχηματίζουν αριστερή στροφή τότε προστίθενται το P στην λίστα με τα σημεία του κυρτού περιβλήματος και ο αλγόριθμος τερματίζει.

Chart, line chart, polygon

Description automatically generated

**Σχήμα 1.7:** Τα σημεία του Κυρτού περιβλήματος

## Υπολογισμός Ελάχιστου Περικλείοντος Κύκλου

Στην ενότητα αυτή θα περιγράψουμε το θεωρητικό υπόβαθρο, κάποια απαραίτητα μαθηματικά καθώς επίσης και παραδείγματα εφαρμογής των αλγορίθμων. Θα εξηγήσουμε αναλυτικά πως ο αλγόριθμος παίρνει ως είσοδο ένα σύνολο τυχαίων σημείων, πως από αυτά επιλέγει μόνο εκείνα που ορίζουν το κυρτό πολύγωνο και έπειτα τον τρόπο με τον οποίο υπολογίζει τον Ελάχιστο Περικλείοντα Κύκλο καθώς και τα Αντίστοιχα Διαγράμματα Voronoi

### Θεωρητικό Υπόβαθρο

### Περιγραφή του Αλγορίθμου

### Παράδειγμα Εφαρμογής του Αλγορίθμου

## Υπολογισμός Διαγράμματος Voronoi

Στον Πίνακα 2.1 βλέπουμε τα στατιστικά για τα διαφορετικά σύνολα δεδομένων: το μέγεθος, τη μέση τιμή της Μετρικής 1, και την μέση τιμή της Μετρικής 2.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Μέγεθος | Μετρική 1 | Μετρική 2 |
| Dataset 1 | 2000 | 2.3 | 23 |
| Dataset 2 | 1000 | 4.3 | 45 |
| Dataset 3 | 3000 | 6.7 | 100 |

Πίνακας .. Στατιστικά για τα δεδομένα

### Περιγραφή του Αλγορίθμου

### Παράδειγμα Εφαρμογής του Αλγορίθμου

## 

# Η Υλοποίηση

## Είσοδος-Έξοδος

Η εφαρμογή έχει την δυνατότητα να διαβάσει ένα σύνολο σημείων το ευκλείδειου επιπέδου από ένα σύνολο αρχείων και να εξάγει τα αντίστοιχα γραφήματα.

Πιο συγκεκριμένα, ο χρήστης μπορεί να φορτώσει τα σημεία του από το παρακάτω σύνολο αρχείων:

* .csv: Csv αρχείο με δύο στήλες αριθμών
* .txt: Txt αρχείο με δύο στήλες αριθμών χωρισμένους με κόμμα μεταξύ του
* .xls: Excel 1997-2003 αρχείο με δύο στήλες αριθμών
* .xlsx: Excel 2007-2022 αρχείο με δύο στήλες αριθμών

Έπειτα, η εφαρμογή να υπολογίσει και θα εμφανίσει σε 3 ξεχωριστά παράθυρα τα αντίστοιχα γραφήματα για το κάθε διάγραμμα.

## Δομές Δεδομένων

Στην εφαρμογή χρησιμοποιούμε κατά βάση 3 Δομές Δεδομένων.

Η πρώτη Δομή είναι ένα σύνολο από πίνακες(Array List) που χρησιμοποιήσαμε για να αποθηκεύσουμε το αρχικό σύνολο σημείων, των σημείων του κυρτού πολυγώνου, των σημείων του κύκλου καθώς. Κύριος σκοπός αυτής της δομής είναι η αποθήκευση και μόνο των αντίστοιχων σημείων και δεν συμμετέχουν σε υπολογισμούς παρά μόνο για ανάγνωση των περιεχομένων τους διότι έχουν πολύ υψηλή πολυπλοκότητα για βασικές λειτουργίες που χρειαστήκαμε κατά την υλοποίηση των αλγορίθμων (όπως O(n) για διαγραφή ενός στοιχείου).

Η δεύτερη Δομή είναι ένα Κοκκινόμαυρο Ισοζυγισμένο Δυαδικό Δένδρο Αναζήτησης(Tree Map) το οποίο χρησιμοποιήσαμε για να αποθηκεύσουμε τα radius για κάθε 3-άδα καθώς επίσης και τον κεντρικό κόμβο-σημείο της 3-άδας. Αυτό μας βοήθησε πολύ διότι για σχεδόν όλες τις υποστηριζόμενες λειτουργίες που χρειαστήκαμε η πολυπλοκότητα είναι O(logn) και επιπλέον έχει την ιδιότητα ότι διατηρεί τα στοιχεία του ταξινομημένα. Αυτό επίσης μας βοήθησε στο γεγονός ότι ο κάθε αλγόριθμος χρειαζόταν σε κάθε επανάληψη να αφαιρεί και έναν κόμβο το οποίο σημαίνει ότι θα έπρεπε να υπολογιστούν ξανά 3 radius(μιας και ένας κόμβος συμμετέχει σε 3 3-άδες) τα οποία και θα έπρεπε να αφαιρεθούν από το δέντρο(πολυπλοκότητα 3\*O(logn)) και να γίνουν 2 νέες εισαγωγές radius στο δέντρο(πολυπλοκότητα 2\*O(logn)) στην σωστή θέση ώστε το δένδρο να παραμείνει ισοζυγισμένο.

Η τρίτη και τελευταία Δομή είναι ένα σύνολο από λεξικά(Hash Map) που τα χρησιμοποιήσαμε για να αποθηκεύσουμε στο πρώτο τον πίνακα γειτνίασης και στο δεύτερο αντίστοιχα radius και τους κεντρικούς κόμβων. Ο λόγος που χρησιμοποιήσαμε λεξικά είναι διότι η πολυπλοκότητα για τις βασικές λειτουργίες που χρειαστήκαμε είναι Ο(1). Αυτό μας βοήθησε αρκετά διότι κάναμε πολλές αναζητήσεις με συγκεκριμένο κλειδί όπως επίσης και διαγραφές.

## Κλάσεις/Πακέτα

Σε αυτήν την ενότητα θα περιγράψουμε περιληπτικά και θα παραθέσουμε κομμάτια(κλάσεις και μεθόδους) του κώδικα. Αξίζει να σημειώσουμε ότι χρησιμοποιήσαμε το πρότυπο αρχιτεκτονικής MVC(Model View Control) για την γενική μορφή του κώδικά καθώς και άλλες μεθόδους δημιουργίας κλάσεων(όπως Factory Pattern) ώστε ο κώδικάς να είναι αναγνώσιμος και επεκτάσιμος από κάποιον άλλον.

Η δομή των παρακάτω κεφαλαίων είναι η εξής: Κλάση/Πακέτο

### MainGUI/View

Η κλάση ‘MainGUI’ είναι υπεύθυνη για την δημιουργία του κύριου παραθύρου καθώς και την αρχικοποίηση των Handlers(Κλάσεις που διαχειρίζονται τις λειτουργίες). Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της. Επιπλέον βρίσκεται και με μέθοδος ‘main’ η οποία είναι υπεύθυνη για την έναρξη του προγράμματος.

//--------------------------------------------------------------------------------------------------------------

**package** View;

**import** java.net.URL;

**import** java.util.ArrayList;

**import** Control.ChangeLanguageButtonHandler;

**import** Control.ExitButtonHandler;

**import** Control.InfoButtonHandler;

**import** Control.LoadFileButtonHandler;

**import** Control.PaperTitleHandler;

**import** Control.PutPointsButtonHandler;

**import** javafx.application.Application;

**import** javafx.event.EventHandler;

**import** javafx.geometry.Pos;

**import** javafx.scene.Cursor;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Button;

**import** javafx.scene.control.Label;

**import** javafx.scene.image.Image;

**import** javafx.scene.image.ImageView;

**import** javafx.scene.layout.BorderPane;

**import** javafx.scene.layout.VBox;

**import** javafx.stage.Stage;

**import** javafx.stage.WindowEvent;

**public** **class** MainGUI **extends** Application

{

**private** String javaVersion = System.*getProperty*("java.version");

**private** String javafxVersion = System.*getProperty*("javafx.version");

**private** String version = "v0.9.9";

**private** Stage stage;

**private** Scene scene;

**private** Button changeLanguageButton;

**private** Button loadFileButton;

**private** Button putPointsButton;

**private** Button exitButton;

**private** Button infoButton;

**private** BorderPane borderPane;

**private** VBox vbox;

**private** Label paperTitle;

**private** ImageView imageViewMain;

**private** ImageView imageViewMain2;

**private** ArrayList<Button> buttons;

@Override

**public** **void** start(Stage stage)

{

buttons = **new** ArrayList<Button>();

**this**.stage = stage;

createButtons();

createPaperTitle();

createSecondPane();

createMainPane();

scene = **new** Scene(borderPane);

scene.getStylesheets().add("styles.css");

createAndSetHandlers();

createMainLogo2();

createStage();

}

**public** **static** **void** main(String[] args)

{

*launch*();

}

**private** **void** createAndSetHandlers()

{

changeLanguageButton.setOnAction(**new** ChangeLanguageButtonHandler(buttons, paperTitle));

loadFileButton.setOnAction(**new** LoadFileButtonHandler());

putPointsButton.setOnAction(**new** PutPointsButtonHandler());

exitButton.setOnAction(**new** ExitButtonHandler(stage));

infoButton.setOnAction(**new** InfoButtonHandler());

}

**private** **void** createButtons()

{

changeLanguageButton = **new** Button();

loadFileButton = **new** Button();

putPointsButton = **new** Button();

exitButton = **new** Button();

infoButton = **new** Button(" ? ");

changeLanguageButton.setId("changeLanguageButton");

loadFileButton.setId("loadFileButton");

putPointsButton.setId("putPointsButton");

exitButton.setId("exitButton");

infoButton.setId("infoButton");

changeLanguageButton.setCursor(Cursor.***HAND***);

loadFileButton.setCursor(Cursor.***HAND***);

putPointsButton.setCursor(Cursor.***HAND***);

exitButton.setCursor(Cursor.***HAND***);

infoButton.setCursor(Cursor.***HAND***);

loadFileButton.setMaxWidth(Double.***MAX\_VALUE***);

putPointsButton.setMaxWidth(Double.***MAX\_VALUE***);

exitButton.setMaxWidth(Double.***MAX\_VALUE***);

buttons.add(changeLanguageButton);

buttons.add(loadFileButton);

buttons.add(putPointsButton);

buttons.add(exitButton);

buttons.add(infoButton);

}

**private** **void** createMainPane()

{

borderPane = **new** BorderPane();

borderPane.setId("borderPane");

createMainLogo();

borderPane.setTop(**null**);

borderPane.setLeft(imageViewMain);

borderPane.*setAlignment*(imageViewMain, Pos.***CENTER***);

//------------------------------------------------------------------

borderPane.setCenter(vbox);

borderPane.*setAlignment*(vbox, Pos.***CENTER***);

//------------------------------------------------------------------

borderPane.setTop(changeLanguageButton);

borderPane.*setAlignment*(changeLanguageButton, Pos.***CENTER\_RIGHT***);

borderPane.setBottom(infoButton);

borderPane.*setAlignment*(infoButton, Pos.***CENTER\_RIGHT***);

}

**private** **void** createPaperTitle()

{

paperTitle = **new** Label();

paperTitle.setCursor(Cursor.***HAND***);

paperTitle.setOnMouseClicked(**new** PaperTitleHandler());

}

**private** **void** createSecondPane()

{

vbox = **new** VBox(10);

vbox.setId("vbox");

vbox.getChildren().addAll(paperTitle, loadFileButton, putPointsButton, exitButton);

}

**private** **void** createMainLogo()

{

URL url = getClass().getResource("/Icons/MainLogo\_W.png");

Image imageMain = **new** Image(url.toString());

imageViewMain = **new** ImageView(imageMain);

imageViewMain.setFitHeight(573);

imageViewMain.setFitWidth(300);

}

**private** **void** createMainLogo2()

{

URL url = getClass().getResource("/Icons/MainLogo\_W.png");

Image imageMain2 = **new** Image(url.toString());

imageViewMain2 = **new** ImageView(imageMain2);

imageViewMain2.setFitHeight(38);

imageViewMain2.setFitWidth(17);

}

**private** **void** createStage()

{

URL url = getClass().getResource("/Icons/circle.png");

Image stageImage = **new** Image(url.toString());

stage.getIcons().add(stageImage);

stage.setTitle("Sven Skyum 1991, JavaFX " + javafxVersion + ", running on Java " + javaVersion +", "+version);

stage.setHeight(700);

stage.setWidth(600);

stage.setResizable(**false**);

stage.setScene(scene);

stage.setOnCloseRequest(**new** EventHandler<WindowEvent>() {

**public** **void** handle(WindowEvent we)

{

stage.close();

System.*exit*(0);

}

});

stage.show();

}

}

//--------------------------------------------------------------------------------------------------------------

### SmallestEnclosingCircleGUI/View

Η κλάση ΄SmallestEnclosingCircleGUI΄ είναι υπεύθυνη για την δημιουργία και την εμφάνιση του διαγράμματος του Ελάχιστου Περικλείοντος Κύκλους στον χρήστη. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** View;

**import** java.awt.BasicStroke;

**import** java.awt.Color;

**import** java.awt.Font;

**import** java.awt.geom.Ellipse2D;

**import** java.net.URL;

**import** java.util.ArrayList;

**import** org.jfree.chart.ChartFactory;

**import** org.jfree.chart.JFreeChart;

**import** org.jfree.chart.annotations.XYShapeAnnotation;

**import** org.jfree.chart.fx.ChartViewer;

**import** org.jfree.chart.plot.XYPlot;

**import** org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;

**import** org.jfree.chart.title.TextTitle;

**import** org.jfree.data.xy.XYSeries;

**import** org.jfree.data.xy.XYSeriesCollection;

**import** Control.CustomMouseListener;

**import** javafx.scene.Scene;

**import** javafx.scene.image.Image;

**import** javafx.stage.Stage;

**import** javafx.geometry.Point2D;

**public** **class** SmallestEnclosingCircleGraphGUI

{

**private** ArrayList<Point2D> allPoints;

**private** ArrayList<Point2D> convexPoints;

**private** ArrayList<Point2D> circlePoints;

**private** Stage circleStage;

**private** String path;

**private** String title;

**private** XYSeriesCollection dataset;

**private** XYPlot plot;

**private** TextTitle textSubTitle;

**private** Ellipse2D circle;

**public** SmallestEnclosingCircleGraphGUI(String path)

{

**this**.path = path;

}

**public** **void** setTitle(String title)

{

**this**.title = title;

}

**public** **void** setAllPoints(ArrayList<Point2D> allPoints)

{

**this**.allPoints = allPoints;

}

**public** **void** setConvexPoints(ArrayList<Point2D> convexPoints)

{

**this**.convexPoints = convexPoints;

}

**public** **void** setCirclePoints(ArrayList<Point2D> circlePoints)

{

**this**.circlePoints = circlePoints;

}

**public** **void** setCircleObject(Ellipse2D circle)

{

**this**.circle = circle;

}

**public** **void** initialize(**int** x, **int** y)

{

createStage(x, y);

displaySmallestEnclosingCircle();

circleStage.show();

}

**private** **void** createStage(**int** x, **int** y)

{

circleStage = **new** Stage();

circleStage.setTitle(title+" -> "+path+" -> "+(allPoints.size())+" points");

circleStage.setHeight(700);

circleStage.setWidth(700);

circleStage.setX(x);

circleStage.setY(y);

circleStage.setResizable(**true**);

URL url = getClass().getResource("/Icons/circle.png");

Image stageImage = **new** Image(url.toString());

circleStage.getIcons().add(stageImage);

}

**private** **void** displaySmallestEnclosingCircle()

{

textSubTitle = **new** TextTitle("Current Point: None");

textSubTitle.setFont(**new** Font("SansSerif", Font.***PLAIN***, 12));

makePlot();

ChartViewer viewer = **new** ChartViewer(makeChart());

viewer.addChartMouseListener(**new** CustomMouseListener(textSubTitle));

circleStage.setScene(**new** Scene(viewer));

}

**private** XYSeriesCollection makeSeriesAndDataset()

{

**int** circlePointsSize = circlePoints.size();

**int** convexPointsSize = convexPoints.size();

**int** allPointsSize = allPoints.size();

XYSeries series1 = **new** XYSeries("allPoints -> "+(allPointsSize-convexPointsSize)+"+convexPoints+circlePoints");

XYSeries series2 = **new** XYSeries("convexPoints -> "+(convexPointsSize - circlePointsSize)+"+circlePoints"+" ||");

XYSeries series3 = **new** XYSeries("circlePoints -> "+circlePointsSize+" ||");

dataset = **new** XYSeriesCollection();

**for**(**int** i=0; i<allPointsSize; i++)

{

series1.add(allPoints.get(i).getX(),allPoints.get(i).getY());

}

**for**(**int** i=0; i<convexPointsSize; i++)

{

series2.add(convexPoints.get(i).getX(), convexPoints.get(i).getY());

}

**for**(**int** i=0; i<circlePointsSize; i++)

{

series3.add(circlePoints.get(i).getX(), circlePoints.get(i).getY());

}

dataset.addSeries(series3);

dataset.addSeries(series2);

dataset.addSeries(series1);

dataset.setAutoWidth(**true**);

**return** dataset;

}

**private** **void** makePlot()

{

JFreeChart scatterPlot = ChartFactory.*createScatterPlot*(

"Current Point: None", // Chart title

"X", // X-Axis Label

"Y", // Y-Axis Label

makeSeriesAndDataset() // Dataset for the Chart

);

plot = (XYPlot)scatterPlot.getPlot();

plot.addAnnotation(makeCircle());

plot.setDomainPannable(**true**);

plot.setRangePannable(**true**);

plot.setDomainCrosshairLockedOnData(**true**);

plot.setRangeMinorGridlinesVisible(**true**);

plot.setDomainCrosshairVisible(**true**);

plot.setDomainZeroBaselineVisible(**true**);

plot.setOutlineVisible(**true**);

plot.setBackgroundPaint(Color.***WHITE***);

plot.setDomainGridlinePaint(Color.***GRAY***);

plot.setRangeGridlinePaint(Color.***GRAY***);

plot.setRenderer(makeRenderer());

}

**private** XYShapeAnnotation makeCircle()

{

XYShapeAnnotation annotation = **new** XYShapeAnnotation(circle, **new** BasicStroke(1.0f), Color.***RED***, **null**);

annotation.setToolTipText("center=("+circle.getCenterX()+", "+circle.getCenterY()+")\nradius="+circle.getWidth()/2);

**return** annotation;

}

**private** XYLineAndShapeRenderer makeRenderer()

{

XYLineAndShapeRenderer renderer2 = **new** XYLineAndShapeRenderer();

renderer2.setSeriesPaint(0, Color.***RED***); //circle points

renderer2.setSeriesLinesVisible(0, **false**);

renderer2.setSeriesShapesVisible(0, **true**);

renderer2.setSeriesPaint(1, Color.***BLUE***); //convex points

renderer2.setSeriesLinesVisible(1, **false**);

renderer2.setSeriesShapesVisible(1, **true**);

renderer2.setSeriesPaint(2, Color.***ORANGE***); // all points

renderer2.setSeriesLinesVisible(2, **false**);

renderer2.setSeriesShapesVisible(2, **true**);

**return** renderer2;

}

**private** JFreeChart makeChart()

{

JFreeChart chart = **new** JFreeChart(plot);

chart.setTitle(title);

chart.getTitle().setPaint(Color.*decode*("#006699"));

chart.getTitle().setFont(**new** Font("Arial", Font.***TRUETYPE\_FONT***, 24));

chart.setBackgroundPaint(Color.*decode*("#f0f9f6"));

chart.addSubtitle(textSubTitle);

chart.setElementHinting(**true**);

chart.setTextAntiAlias(**true**);

chart.setNotify(**true**);

chart.setAntiAlias(**true**);

chart.setBorderVisible(**false**);

**return** chart;

}

}

//--------------------------------------------------------------------------------------------------------------

### VoronoiGraphGUI/View

Η κλάση ΄VoronoiGraphGUI΄ είναι υπεύθυνη για την δημιουργία και την εμφάνιση των διαγραμμάτων Voronoi(κοντινότερου και μακρύτερου γείτονα) στον χρήστη. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** View;

**import** java.awt.BasicStroke;

**import** java.awt.Color;

**import** java.awt.Font;

**import** java.net.URL;

**import** java.util.ArrayList;

**import** org.jfree.chart.ChartFactory;

**import** org.jfree.chart.JFreeChart;

**import** org.jfree.chart.fx.ChartViewer;

**import** org.jfree.chart.plot.XYPlot;

**import** org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;

**import** org.jfree.chart.title.TextTitle;

**import** org.jfree.data.xy.XYSeries;

**import** org.jfree.data.xy.XYSeriesCollection;

**import** Control.CustomMouseListener;

**import** javafx.geometry.Point2D;

**import** javafx.scene.Scene;

**import** javafx.scene.image.Image;

**import** javafx.stage.Stage;

**public** **class** VoronoiGraphGUI

{

**private** **int** allPointsSize;

**private** ArrayList<Point2D> convexPoints;

**private** ArrayList<Point2D> K;

**private** ArrayList<ArrayList<Point2D>> E;

**private** Stage voronoiStage;

**private** String path;

**private** String title;

**private** XYPlot plot;

**private** TextTitle textSubTitle;

**public** VoronoiGraphGUI(String path)

{

**this**.path = path;

}

**public** **void** setTitle(String title)

{

**this**.title = title;

}

**public** **void** setAllPointsSize(**int** allPointsSize)

{

**this**.allPointsSize = allPointsSize;

}

**public** **void** setConvexPoints(ArrayList<Point2D> convexPoints)

{

**this**.convexPoints = convexPoints;

}

**public** **void** setK(ArrayList<Point2D> K)

{

**this**.K = K;

}

**public** **void** setE(ArrayList<ArrayList<Point2D>> E)

{

**this**.E = E;

}

**public** **void** initialize(**int** x, **int** y)

{

createStage(x, y);

displaySmallestEnclosingCircle();

voronoiStage.show();

}

**private** **void** createStage(**int** x, **int** y)

{

voronoiStage = **new** Stage();

voronoiStage.setTitle(title+" -> "+path+" -> "+(allPointsSize)+" points");

voronoiStage.setHeight(700);

voronoiStage.setWidth(700);

voronoiStage.setX(x);

voronoiStage.setY(y);

voronoiStage.setResizable(**true**);

URL url = getClass().getResource("/Icons/circle.png");

Image stageImage = **new** Image(url.toString());

voronoiStage.getIcons().add(stageImage);

}

**private** **void** displaySmallestEnclosingCircle()

{

textSubTitle = **new** TextTitle("Current Point: None");

textSubTitle.setFont(**new** Font("SansSerif", Font.***PLAIN***, 12));

makePlot();

ChartViewer viewer = **new** ChartViewer(makeChart());

viewer.addChartMouseListener(**new** CustomMouseListener(textSubTitle));

voronoiStage.setScene(**new** Scene(viewer));

}

**private** XYSeriesCollection makeSeriesAndDataset()

{

XYSeries series0 = **new** XYSeries("convexPoints -> "+(convexPoints.size()));

XYSeries series1 = **new** XYSeries("Voronoi Edges -> "+(K.size())+" ||");

XYSeries series2 = **new** XYSeries("Voronoi Lines -> "+(E.size()+" ||"));

XYSeriesCollection dataset = **new** XYSeriesCollection();

**for**(**int** i=0; i<convexPoints.size(); i++)

{

series0.add(convexPoints.get(i).getX(), convexPoints.get(i).getY());

}

**for**(**int** i=0; i<K.size(); i++)

{

series1.add(K.get(i).getX(), K.get(i).getY());

}

dataset.addSeries(series2); // Voronoi Cell

dataset.addSeries(series1); // Voronoi Edges

dataset.addSeries(series0); // convexPoints

**for**(**int** i=0; i<E.size(); i++) // Voronoi Cells

{

XYSeries series = **new** XYSeries(i+"");

series.add(E.get(i).get(0).getX(), E.get(i).get(0).getY());

series.add(E.get(i).get(1).getX(), E.get(i).get(1).getY());

dataset.addSeries(series);

}

dataset.setAutoWidth(**true**);

**return** dataset;

}

**private** **void** makePlot()

{

JFreeChart scatterPlot = ChartFactory.*createScatterPlot*(

"Current Point: None", // Chart title

"X", // X-Axis Label

"Y", // Y-Axis Label

makeSeriesAndDataset() // Dataset for the Chart

);

plot = (XYPlot)scatterPlot.getPlot();

plot.setDomainPannable(**true**);

plot.setRangePannable(**true**);

plot.setDomainCrosshairLockedOnData(**true**);

plot.setRangeMinorGridlinesVisible(**true**);

plot.setDomainCrosshairVisible(**true**);

plot.setDomainZeroBaselineVisible(**true**);

plot.setOutlineVisible(**true**);

plot.setBackgroundPaint(Color.***WHITE***);

plot.setDomainGridlinePaint(Color.***GRAY***);

plot.setRangeGridlinePaint(Color.***GRAY***);

plot.setRenderer(makeRenderer());

}

**private** XYLineAndShapeRenderer makeRenderer()

{

XYLineAndShapeRenderer renderer2 = **new** XYLineAndShapeRenderer();

renderer2.setSeriesPaint(2, Color.***BLUE***); //convex points

renderer2.setSeriesLinesVisible(2, **false**);

renderer2.setSeriesShapesVisible(2, **true**);

renderer2.setSeriesPaint(1, Color.***RED***); //voronoi edges

renderer2.setSeriesLinesVisible(1, **false**);

renderer2.setSeriesShapesVisible(1, **true**);

renderer2.setSeriesPaint(0, Color.***PINK***); //voronoi cells

renderer2.setSeriesLinesVisible(0, **false**);

renderer2.setSeriesShapesVisible(0, **false**);

**for**(**int** i=0; i<E.size(); i++)

{

**int** offset = 3;

renderer2.setSeriesPaint(offset+i, Color.*decode*("#8fbc8f")); //voronoi cells

renderer2.setSeriesLinesVisible(offset+i, **true**);

renderer2.setSeriesShapesVisible(offset+i, **false**);

renderer2.setSeriesStroke(offset+i, **new** BasicStroke(2.0f)); //light green

renderer2.setSeriesItemLabelsVisible(offset+i, **false**);

renderer2.setDefaultItemLabelsVisible(**false**);

renderer2.setSeriesVisibleInLegend(offset+i, **false**);

}

**return** renderer2;

}

**private** JFreeChart makeChart()

{

JFreeChart chart = **new** JFreeChart(plot);

chart.setTitle(title);

//=======================================================

chart.getTitle().setPaint(Color.*decode*("#006699"));

chart.getTitle().setFont(**new** Font("Arial", Font.***TRUETYPE\_FONT***, 24));

chart.setBackgroundPaint(Color.*decode*("#f0f9f6"));

//=======================================================

chart.addSubtitle(textSubTitle);

chart.setElementHinting(**true**);

chart.setTextAntiAlias(**true**);

chart.setNotify(**true**);

chart.setAntiAlias(**true**);

chart.setBorderVisible(**false**);

**return** chart;

}

}

//--------------------------------------------------------------------------------------------------------------

### AlgorithmsHandler/Control

Η κλάση ΄AlgorithmsHandler΄ είναι υπεύθυνη για την δημιουργία και την έναρξη όλων των αλγορίθμων(GrahamScan, SmallestEnclosingCircle, Voronoi). Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Control;

**import** Model.SmallestEnclosingCirlce;

**import** Model.NeighborVoronoiDiagram;

**import** Model.DataBase;

**import** Model.GrahamScan;

**public** **class** AlgorithmsHandler

{

**private** GrahamScan grahamScan;

**private** SmallestEnclosingCirlce algorithm1;

**private** NeighborVoronoiDiagram nearestNeighbor;

**private** NeighborVoronoiDiagram farthestNeighbor;

**private** DataBase dataBase;

**public** AlgorithmsHandler()

{

dataBase = DataBase.*getInstance*();

dataBase.initializeKandE();

callAlgorithms();

}

**private** **void** callAlgorithms()

{

callGrahamScan();

callSmallestEnclosingCircle();

callNearestNeighborVoronoi();

callFarthestNeighborVoronoi();

}

**private** **void** callGrahamScan()

{

grahamScan = **new** GrahamScan();

grahamScan.initialize();

}

**private** **void** callSmallestEnclosingCircle()

{

algorithm1 = **new** SmallestEnclosingCirlce();

dataBase.prepareForAlgorithm(0);

algorithm1.computeSmallestEnclosingCircle();

}

**private** **void** callNearestNeighborVoronoi()

{

nearestNeighbor = **new** NeighborVoronoiDiagram();

dataBase.prepareForAlgorithm(1);

nearestNeighbor.computeVoronoiDiagram();

}

**private** **void** callFarthestNeighborVoronoi()

{

farthestNeighbor = **new** NeighborVoronoiDiagram();

dataBase.prepareForAlgorithm(2);

farthestNeighbor.computeVoronoiDiagram();

}

}

//--------------------------------------------------------------------------------------------------------------

### ChangeLanguageButtonHandler/Control

Η κλάση ΄ChangeLanguageButtonHandler΄ είναι υπεύθυνη για την εναλλαγή της γλώσσας του προγράμματος μεταξύ Ελληνικών-Αγγλικών. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Control;

**import** java.util.ArrayList;

**import** Model.TextHandler;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.scene.control.Button;

**import** javafx.scene.control.Label;

**public** **class** ChangeLanguageButtonHandler **implements** EventHandler<ActionEvent>

{

**private** TextHandler textHandler;

**private** Boolean flag=**true**;

**private** Button changeLanguageButton;

**private** Label paperTitle;

**private** Button loadFileButton;

**private** Button putPointsButton;

**private** Button exitButton;

**public** ChangeLanguageButtonHandler(ArrayList<Button> buttons, Label paperTitle)

{

textHandler = TextHandler.*getInstance*();

**this**.paperTitle = paperTitle;

changeLanguageButton = buttons.get(0);

loadFileButton = buttons.get(1);

putPointsButton = buttons.get(2);

exitButton = buttons.get(3);

//first time

setTexts();

}

@Override

**public** **void** handle(ActionEvent arg0)

{

//flag=true -> EN

//flag=false -> GR

flag=!flag;

textHandler = TextHandler.*getInstance*();

textHandler.setLanguage(flag? "EN" : "GR");

setTexts();

}

**private** **void** setTexts()

{

changeLanguageButton.setText(textHandler.getChangeLanguageButtonText());

paperTitle.setText(textHandler.getPaperTitleText());

loadFileButton.setText(textHandler.getLoadFileButtonText());

putPointsButton.setText(textHandler.getPutPointsButtonText());

exitButton.setText(textHandler.getExitButtonText());

}

}

//--------------------------------------------------------------------------------------------------------------

### CustomMouseListener/Control

Η κλάση ΄CustomMouseListener΄ είναι υπεύθυνη για την εμφάνιση των συντεταγμένων ενός σημείου καθώς και επιπλέον πληροφοριών πάνω στα διαγράμματα όταν ο χρήστη αλληλεπιδρά με αυτά μέσω του ποντικιού. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Control;

**import** org.jfree.chart.fx.interaction.ChartMouseEventFX;

**import** org.jfree.chart.fx.interaction.ChartMouseListenerFX;

**import** org.jfree.chart.title.TextTitle;

**public** **class** CustomMouseListener **implements** ChartMouseListenerFX

{

**private** TextTitle textTitle;

**private** **double** x;

**private** **double** y;

**public** CustomMouseListener(TextTitle textTitle)

{

**this**.textTitle = textTitle;

}

@Override

**public** **void** chartMouseClicked(ChartMouseEventFX arg0)

{

x = arg0.getChart().getXYPlot().getDomainCrosshairValue();

y = arg0.getChart().getXYPlot().getRangeCrosshairValue();

textTitle.setText("Current Point: ("+x+"1, "+y+")");

}

@Override

**public** **void** chartMouseMoved(ChartMouseEventFX arg0)

{

}

}

//--------------------------------------------------------------------------------------------------------------

### ExitButtonHandler/Control

Η κλάση ΄ExitButtonHandler΄ είναι υπεύθυνη για τον τερματισμό του προγράμματος καθώς και όλων των διεργασιών του όταν εκείνος κάνει κλικ στο κουμπί ‘Exit’-‘Έξοδος’. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Control;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.stage.Stage;

**public** **class** ExitButtonHandler **implements** EventHandler<ActionEvent>

{

**private** Stage stage;

**public** ExitButtonHandler(Stage stage)

{

**this**.stage = stage;

}

@Override

**public** **void** handle(ActionEvent arg0)

{

stage.close();

System.*exit*(0);

}

}

//--------------------------------------------------------------------------------------------------------------

### GraphGUIHandler/Control

Η κλάση ΄GraphGUIHandlerHandler΄ είναι υπεύθυνη για την δημιουργία και την έναρξη όλων των διαγραμμάτων καθώς και να περάσει τις απαραίτητες πληροφορίες σε αυτά(π.χ. σημεία κυρτού πολυγώνου κτλ.). Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Control;

**import** Model.DataBase;

**import** View.SmallestEnclosingCircleGraphGUI;

**import** View.VoronoiGraphGUI;

**public** **class** GraphGUIHandler

{

**private** String path;

**private** DataBase dataBase;

**private** SmallestEnclosingCircleGraphGUI circleGraphGUI;

**private** VoronoiGraphGUI farthestNeighborVoronoiGraphGUI;

**public** GraphGUIHandler(String path)

{

**this**.path = path;

dataBase = DataBase.*getInstance*();

makeCircleGraph("Smallest Enclosing Circle", 0, 0);

makeVoronoiGraph("Nearest Neighbor Voronoi Diagram", 700, 0, 0);

makeVoronoiGraph("Farthest Neighbor Voronoi Diagram", 1400, 0, 1);

}

**private** **void** makeCircleGraph(String title, **int** x, **int** y)

{

circleGraphGUI = **new** SmallestEnclosingCircleGraphGUI(path);

circleGraphGUI.setTitle(title);

circleGraphGUI.setAllPoints(dataBase.getAllPoints());

circleGraphGUI.setConvexPoints(dataBase.getConvexPoints());

circleGraphGUI.setCirclePoints(dataBase.getCirclePoints());

circleGraphGUI.setCircleObject(dataBase.getCircleShape());

circleGraphGUI.initialize(x, y);

}

**private** **void** makeVoronoiGraph(String title, **int** x, **int** y, **int** mode)

{

farthestNeighborVoronoiGraphGUI = **new** VoronoiGraphGUI(path);

farthestNeighborVoronoiGraphGUI.setTitle(title);

farthestNeighborVoronoiGraphGUI.setAllPointsSize(dataBase.getAllPointsSize());

farthestNeighborVoronoiGraphGUI.setConvexPoints(dataBase.getConvexPoints());

farthestNeighborVoronoiGraphGUI.setK(dataBase.getK(mode));

farthestNeighborVoronoiGraphGUI.setE(dataBase.getE(mode));

farthestNeighborVoronoiGraphGUI.initialize(x, y);

}

}

//--------------------------------------------------------------------------------------------------------------

### InfoButtonHandler/Control

Η κλάση ‘InfoButtonHandler’ είναι υπεύθυνη για την εμφάνιση του παραθύρου με απαραίτητες πληροφορίες για το πρόγραμμα όταν ο χρήστης κάνει κλικ στο κουμπί ‘?’. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Control;

**import** java.net.URL;

**import** Model.TextHandler;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.geometry.Insets;

**import** javafx.scene.Cursor;

**import** javafx.scene.Scene;

**import** javafx.scene.control.Label;

**import** javafx.scene.image.Image;

**import** javafx.scene.image.ImageView;

**import** javafx.scene.layout.HBox;

**import** javafx.scene.layout.VBox;

**import** javafx.stage.Modality;

**import** javafx.stage.Stage;

**public** **class** InfoButtonHandler **implements** EventHandler<ActionEvent>

{

**private** Stage stage;

**private** VBox layout;

**private** Scene scene;

**private** Label label;

**private** ImageView circle;

**private** ImageView nearestVoronoi;

**private** ImageView farthestVoronoi;

**private** TextHandler textHandler;

@Override

**public** **void** handle(ActionEvent arg0)

{

textHandler = TextHandler.*getInstance*();

createAndShowInfoWindow();

}

**private** **void** createAndShowInfoWindow()

{

stage = **new** Stage();

stage.setTitle(textHandler.getInfoWindowTitle());

URL url = getClass().getResource("/Icons/circle.png");

Image stageImage = **new** Image(url.toString());

stage.getIcons().add(stageImage);

createLabelContent();

createGrahpImages(200);

createBox();

scene = **new** Scene(layout, 640, 700);

scene.getStylesheets().add("styles.css");

stage.setScene(scene);

stage.initModality(Modality.***APPLICATION\_MODAL***);

stage.setResizable(**false**);

stage.show();

}

**private** **void** createLabelContent()

{

label = **new** Label(textHandler.getInfoContent());

label.setId("infoLabel");

label.setCursor(Cursor.***HAND***);

label.setOnMouseClicked(**new** InfoContentHandler());

}

**private** **void** createBox()

{

layout = **new** VBox(10);

HBox hbox = **new** HBox(10);

hbox.setPadding(**new** Insets(10, 10, 10, 10));

hbox.getChildren().addAll(circle, nearestVoronoi, farthestVoronoi);

layout.getChildren().addAll(label, hbox);

}

**private** **void** createGrahpImages(**int** size)

{

createCircleImage(size);

createNearestVoronoiImage(size);

createFarthestVoronoiImage(size) ;

}

**private** **void** createCircleImage(**int** size)

{

URL url0 = getClass().getResource("/Icons/circle (2).png");

Image circleImage = **new** Image(url0.toString());

circle = **new** ImageView(circleImage);

circle.setFitHeight(size);

circle.setFitWidth(size);

}

**private** **void** createNearestVoronoiImage(**int** size)

{

URL url1 = getClass().getResource("/Icons/nearestVoronoi.png");

Image nearestVoronoiImage = **new** Image(url1.toString());

nearestVoronoi = **new** ImageView(nearestVoronoiImage);

nearestVoronoi.setFitHeight(size);

nearestVoronoi.setFitWidth(size);

}

**private** **void** createFarthestVoronoiImage(**int** size)

{

URL url2 = getClass().getResource("/Icons/farthestVoronoi.png");

Image farthestVoronoiImage = **new** Image(url2.toString());

farthestVoronoi = **new** ImageView(farthestVoronoiImage);

farthestVoronoi.setFitHeight(size);

farthestVoronoi.setFitWidth(size);

}

}

//--------------------------------------------------------------------------------------------------------------

### InfoContentHandler/Control

Η κλάση ‘InfoContentHandler’ είναι υπεύθυνη για την εμφάνιση του αποθετηρίου στο οποίο βρίσκεται ο κώδικας του προγράμματος όταν ο χρήστης κάνει κλικ πάνω σε οποιοδήποτε σημείο του κειμένου πληροφοριών. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Control;

**import** java.awt.Desktop;

**import** java.io.IOException;

**import** java.net.URI;

**import** java.net.URISyntaxException;

**import** javafx.event.EventHandler;

**import** javafx.scene.input.MouseEvent;

**public** **class** InfoContentHandler **implements** EventHandler<MouseEvent>

{

**private** String paperURL = "https://github.com/johnprif/Thesis";

@Override

**public** **void** handle(MouseEvent arg0)

{

**try** {

Desktop.*getDesktop*().browse(**new** URI(paperURL));

} **catch** (IOException | URISyntaxException e) {

e.printStackTrace();

}

}

}

//--------------------------------------------------------------------------------------------------------------

### LoadFileButtonHandler/Control

Η κλάση ‘LoadFileButtonHandler’ είναι υπεύθυνη για την εμφάνιση του παραθύρου εξερεύνησης του λειτουργικού συστήματος δίνοντας έτσι την δυνατότητα στον χρήστη να μπορεί να φορτώσει το αρχείο με τα σημεία που επιθυμεί και δίνει τις απαραίτητες πληροφορίες στις υπόλοιπες κλάσεις. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Control;

**import** java.io.File;

**import** Model.TextHandler;

**import** Model\_Loaders.FileLoaderFactory;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.stage.FileChooser;

**import** javafx.stage.Stage;

**public** **class** LoadFileButtonHandler **implements** EventHandler<ActionEvent>

{

**private** Stage fileChooserStage;

**private** String path;

**private** TextHandler textHandler;

**private** FileChooser fileChooser;

**private** FileLoaderFactory fileFactoryLoader;

**private** AlgorithmsHandler algorithmsHandler;

**private** GraphGUIHandler graphGUIHandler;

**public** LoadFileButtonHandler()

{

textHandler = TextHandler.*getInstance*();

}

@Override

**public** **void** handle(ActionEvent arg0)

{

fileChooserStage = **new** Stage();

fileChooser();

}

**private** **void** fileChooser()

{

fileChooser = **new** FileChooser();

fileChooser.setTitle(textHandler.getFileChooserTitleText());

fileChooser.getExtensionFilters().addAll(

**new** FileChooser.ExtensionFilter("CSV", "\*.csv")

,**new** FileChooser.ExtensionFilter("txt", "\*.txt")

,**new** FileChooser.ExtensionFilter("Excel 2007-2022", "\*.xlsx")

,**new** FileChooser.ExtensionFilter("Excel 1997-2003", "\*.xls")

);

File selectedFile = fileChooser.showOpenDialog(fileChooserStage);

**if**(selectedFile != **null**)//pressed OK

{

path = selectedFile.getAbsolutePath();

nextSteps();

}**else**//pressed Cancel

{

}

}

**private** **void** nextSteps()

{

fileFactoryLoader = **new** FileLoaderFactory(path);

fileFactoryLoader.getAllPoints();

algorithmsHandler = **new** AlgorithmsHandler();

graphGUIHandler = **new** GraphGUIHandler(path);

}

}

//--------------------------------------------------------------------------------------------------------------

### PaperTitleHandler/Control

Η κλάση ‘PaperTitleHandler’ είναι υπεύθυνη για την εμφάνιση του αποθετηρίου στο οποίο βρίσκεται η δημοσίευση του Paper(Sven Skyum 1991) όταν ο χρήστης κάνει κλικ πάνω σε οποιοδήποτε σημείο του τίτλου. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Control;

**import** java.awt.Desktop;

**import** java.io.IOException;

**import** java.net.URI;

**import** java.net.URISyntaxException;

**import** javafx.event.EventHandler;

**import** javafx.scene.input.MouseEvent;

**public** **class** PaperTitleHandler **implements** EventHandler<MouseEvent>

{

**private** String paperURL = "https://reader.elsevier.com/reader/sd/pii/002001909190030L?token=792B047EFD2FBA85B976C2FCB728380392A08F2C78D489ECFB26B2F3CE023E7512BEC247BC14F3EBE21A61E69DDFFC54&originRegion=eu-west-1&originCreation=20230129211433";

@Override

**public** **void** handle(MouseEvent arg0)

{

**try** {

Desktop.*getDesktop*().browse(**new** URI(paperURL));

} **catch** (IOException | URISyntaxException e) {

e.printStackTrace();

}

}

}

//--------------------------------------------------------------------------------------------------------------

### PutPointsButtonHandler/Control

ΥΠΟ-ΚΑΤΑΣΚΕΥΗ!

ΜΕΛΛΟΝΤΙΚΟ ΣΕΝΑΡΙΟ

Η κλάση ‘PutPointsButtonHandler’ είναι υπεύθυνη για την εμφάνιση ενός παραθύρου στο οποίο μπορεί ο χρήστης να τοποθετήσει τα δικά του σημεία, κάνοντας κλικ πάνω στο κουμπί ‘Create Points’-‘Δημιουργία Σημείων’. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

ΤΩΡΙΝΗ ΚΑΤΑΣΤΑΣΗ

Η κλάση ‘PutPointsButtonHandler’ είναι υπεύθυνη για την εμφάνιση ενός παραθύρου με το οποίο ενημερώνει τον χρήστη ότι η λειτουργία αυτή δεν είναι ακόμα διαθέσιμη και βρίσκεται υπό ανάπτυξη, κάνοντας κλικ πάνω στο κουμπί ‘Create Points’-‘Δημιουργία Σημείων’. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Control;

**import** Model.TextHandler;

**import** javafx.event.ActionEvent;

**import** javafx.event.EventHandler;

**import** javafx.scene.control.Alert;

**import** javafx.scene.control.Alert.AlertType;

**public** **class** PutPointsButtonHandler **implements** EventHandler<ActionEvent>

{

**private** TextHandler textHandler;

**public** PutPointsButtonHandler()

{

textHandler = TextHandler.*getInstance*();

}

@Override

**public** **void** handle(ActionEvent arg0)

{

notImpementPopup();

}

**public** **void** notImpementPopup()

{

Alert a = **new** Alert(AlertType.***INFORMATION***);

a.setTitle(textHandler.getNotImpementPopupTitle());

a.setHeaderText(textHandler.getNotImpementPopupHeaderText());

a.setContentText(textHandler.getNotImpementContentText());

a.show();

}

}

//--------------------------------------------------------------------------------------------------------------

### DataBase/Model

Η κλάση ‘DataBase’ είναι υπεύθυνη για όλους τους υπολογισμούς που χρειαζόμαστε για τα διαγράμματα καθώς και για την αποδοτική τους αποθήκευση ώστε να μπορούν να γίνονται γρήγορα όλοι οι υπολογισμοί χωρίς να ξεπερνάει κανένας το χρονικό όριο O(logn). Βρίσκονται όλες οι συναρτήσεις που αφορούν τους μαθηματικούς υπολογισμούς(πχ εύρεση της ακτίνας που περνάει από τρία σημεία) καθώς και τις δομές που αναφέραμε στο προηγούμενο υποκεφάλαιο [3.2]. Οπότε, αποτελεί επιπλεόν και την βάση δεδομένων όπου οι αλγόριθμοι διαβάζουν-γράφουν δεδομένα καθώς επίσης αντλούν τα δεδομένα τους και τα γραφήματα. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Model;

**import** java.awt.geom.Ellipse2D;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.TreeMap;

**import** javafx.geometry.Point2D;

**public** **class** DataBase

{

**private** ArrayList<Point2D> allPoints;

**private** ArrayList<Point2D> convexPoints;

**private** ArrayList<Point2D> circlePoints;

**private** **static** **final** DataBase ***instance*** = **new** DataBase();

**private** **int** mode;

**private** Ellipse2D circle;

**private** ArrayList<Point2D> K;

**private** ArrayList<ArrayList<Point2D>> E;

**private** HashMap<Point2D, Point2D> u\_p;

**private** HashMap<Point2D, ArrayList<Point2D>> neighbours;

**private** TreeMap<Double, Point2D> radiusForEachNode;

**private** HashMap<Double, Point2D> existingRadius;

**private** Point2D currCustom;

**private** Point2D prevCustom;

**private** Point2D nextCustom;

**private** **double** maxAngleCustom;

**private** **double** maxRadiusCustom;

**private** ArrayList<ArrayList<Point2D>> allK;

**private** ArrayList<ArrayList<ArrayList<Point2D>>> allE;

**private** DataBase()

{

}

**public** **static** DataBase getInstance()

{

**return** ***instance***;

}

**public** ArrayList<Point2D> getAllPoints()

{

**return** allPoints;

}

**public** ArrayList<Point2D> getConvexPoints()

{

**return** convexPoints;

}

**public** ArrayList<Point2D> getCirclePoints()

{

**return** circlePoints;

}

**public** ArrayList<Point2D> getK(**int** mode)

{

**return** allK.get(mode);

}

**public** ArrayList<ArrayList<Point2D>> getE(**int** mode)

{

**return** allE.get(mode);

}

**public** **void** setAllPoints(ArrayList<Point2D> allPoints)

{

**this**.allPoints = **new** ArrayList<Point2D>(allPoints);

}

**public** **void** setConvexPoints(ArrayList<Point2D> convexPoints)

{

**this**.convexPoints = **new** ArrayList<Point2D>(convexPoints);

}

**public** **void** setCirclePoints(ArrayList<Point2D> circlePoints)

{

**this**.circlePoints = **new** ArrayList<Point2D>(circlePoints);

}

**public** **void** prepareForAlgorithm(**int** mode)

{

**this**.mode = mode;

loadMaps();

**if**(mode == 1 || mode == 2)

{

circle = getCircleShape();

}

}

// ----------O(nlog(n))---------------

**private** **void** loadMaps()

{

**int** size = getConvexPointsSize();

**double** radius;

neighbours = **new** HashMap<Point2D, ArrayList<Point2D>>();

radiusForEachNode = **new** TreeMap<>();

existingRadius = **new** HashMap<Double, Point2D>();

**for** (**int** i = 0; i < size; i++)

{

Point2D prev = convexPoints.get((i + size - 1) % size); //O(1)

Point2D curr = convexPoints.get(i); //O(1)

Point2D next = convexPoints.get((i + 1) % size); //O(1)

radius = getRadius(prev, curr, next); //O(1)

ArrayList<Point2D> kati = **new** ArrayList<Point2D>();

kati.add(prev);

kati.add(next);

neighbours.put(curr, kati); //O(1)

radiusForEachNode.put(radius, curr); //O(log(n))

existingRadius.put(radius, curr);

}

}

//----------O(log(n)) for each calling---------------

**public** Point2D findMaxP()

{

**int** size = neighbours.size(); //O(1)

**if** (size <= 2)

{

**return** **new** Point2D(-1, -1);

}

**if**(mode == 0 || mode == 2) //circle or nearest

{

maxRadiusCustom = radiusForEachNode.lastKey(); //O(1)

}**else** //mode == 1 farthest

{

maxRadiusCustom = radiusForEachNode.firstKey(); //O(1)

}

currCustom = radiusForEachNode.get(maxRadiusCustom); //O(log(n))

prevCustom = neighbours.get(currCustom).get(0); //O(1)

nextCustom = neighbours.get(currCustom).get(1); //O(1)

maxAngleCustom = getAngle(prevCustom, currCustom, nextCustom); //O(1)

**if**(mode == 0)

{

**return** **new** Point2D(maxAngleCustom, maxAngleCustom);

}

**return** currCustom;

}

//----------5\*O(log(n)) for each calling---------------

**public** **void** deleteMaxP()

{

ArrayList<Point2D> left = **new** ArrayList<Point2D>();

ArrayList<Point2D> right = **new** ArrayList<Point2D>();

//---I found that has changed

Point2D leftCurr = neighbours.get(currCustom).get(0);

Point2D leftPrev = neighbours.get(leftCurr).get(0);

Point2D leftNext = neighbours.get(leftCurr).get(1);

Point2D rightCurr = neighbours.get(currCustom).get(1); //O(1)

Point2D rightPrev = neighbours.get(rightCurr).get(0); //O(1)

Point2D rightNext = neighbours.get(rightCurr).get(1); //O(1)

radiusForEachNode.remove(maxRadiusCustom); //O(log(n))

radiusForEachNode.remove(getRadius(leftPrev, leftCurr, leftNext)); //O(log(n))

radiusForEachNode.remove(getRadius(rightPrev, rightCurr, rightNext)); //O(log(n))

existingRadius.remove(maxRadiusCustom);//O(1)

existingRadius.remove(getRadius(leftPrev, leftCurr, leftNext)); //O(1)

existingRadius.remove(getRadius(rightPrev, rightCurr, rightNext)); //O(1))

neighbours.remove(leftCurr); //O(1)

neighbours.remove(currCustom); //O(1)

neighbours.remove(rightCurr); //O(1)

left.add(leftPrev); //O(1)

left.add(rightCurr); //O(1)

right.add(leftCurr); //O(1)

right.add(rightNext); //O(1)

neighbours.put(leftCurr, left); //O(1)

neighbours.put(rightCurr, right); //O(1)

**double** beforeRadius = getRadius(leftPrev, leftCurr, rightCurr);

**double** nextRadius = getRadius(leftCurr, rightCurr, rightNext);

Point2D currPoint;

**if**(!existingRadius.containsKey(beforeRadius)) //O(1)

{

radiusForEachNode.put(beforeRadius, leftCurr); //O(log(n))

existingRadius.put(beforeRadius, leftCurr);//O(1)

}**else**

{

currPoint = existingRadius.get(beforeRadius);//O(1)

**if**(getAngle(neighbours.get(leftCurr).get(0), leftCurr, neighbours.get(leftCurr).get(1)) >= getAngle(neighbours.get(currPoint).get(0), currPoint, neighbours.get(currPoint).get(1)))

{

radiusForEachNode.put(getRadius(neighbours.get(currPoint).get(0), currPoint, neighbours.get(currPoint).get(1)), leftCurr); //O(log(n))

existingRadius.put(getRadius(neighbours.get(currPoint).get(0), currPoint, neighbours.get(currPoint).get(1)), leftCurr); //O(1)

}

}

**if**(!existingRadius.containsKey(nextRadius)) //O(1)

{

radiusForEachNode.put(nextRadius, rightCurr); //O(log(n))

existingRadius.put(nextRadius, rightCurr);//O(1)

}**else**

{

currPoint = existingRadius.get(nextRadius);//O(1)

**if**(getAngle(neighbours.get(rightCurr).get(0), rightCurr, neighbours.get(rightCurr).get(1)) >= getAngle(neighbours.get(currPoint).get(0), currPoint, neighbours.get(currPoint).get(1)))

{

radiusForEachNode.put(getRadius(neighbours.get(currPoint).get(0), currPoint, neighbours.get(currPoint).get(1)), rightCurr); //O(log(n))

existingRadius.put(getRadius(neighbours.get(currPoint).get(0), currPoint, neighbours.get(currPoint).get(1)), rightCurr); //O(1)

}

}

}

**public** **int** getHashCirclePointsSize()

{

**return** neighbours.size();

}

**public** **void** moveToCircleArray()

{

HashMap<Point2D, Point2D> neighboursTemp = **new** HashMap<Point2D, Point2D>();

**for** (Point2D key : neighbours.keySet())

{

neighboursTemp.put(key, key);

}

circlePoints = **new** ArrayList<Point2D>(neighboursTemp.values());

}

**public** Point2D getPrev()

{

**return** prevCustom;

}

**public** Point2D getNext()

{

**return** nextCustom;

}

**public** **int** getAllPointsSize()

{

**return** allPoints.size();

}

**public** **int** getConvexPointsSize()

{

**return** convexPoints.size();

}

**public** **int** getCirclePointsSize()

{

**return** circlePoints.size();

}

**public** Ellipse2D getCircleShape()

{

**int** n = getCirclePointsSize();

Point2D point1 = circlePoints.get(0);

Point2D point2 = circlePoints.get(1);

Point2D point3 = **null**;

**if** (n == 3)

{

point3 = circlePoints.get(2);

}

circle = **new** Ellipse2D.Double(getCenter(point1, point2, point3).getX() - getRadius(point1, point2, point3), getCenter(point1, point2, point3).getY() - getRadius(point1, point2, point3), getRadius(point1, point2, point3) \* 2, getRadius(point1, point2, point3) \* 2);

**return** circle;

}

**public** **double** getRadius(Point2D p, Point2D q, Point2D r)

{

**double** x1 = p.getX();

**double** y1 = p.getY();

**double** x2 = q.getX();

**double** y2 = q.getY();

**double** x3;

**double** y3;

**double** R = 0;

**if**(r == **null**)

{

R = p.distance(q) / 2;

**return** R;

}

**if** ((!p.equals(q)) && (!q.equals(r)) && (!p.equals(r)))

{

//----------------------

x3 = r.getX();

y3 = r.getY();

//----------------------

**double** x12 = x1 - x2;

**double** x13 = x1 - x3;

**double** y12 = y1 - y2;

**double** y13 = y1 - y3;

**double** y31 = y3 - y1;

**double** y21 = y2 - y1;

**double** x31 = x3 - x1;

**double** x21 = x2 - x1;

// x1^2 - x3^2

**double** sx13 = x1 \* x1 - x3 \* x3;

// y1^2 - y3^2

**double** sy13 = y1 \* y1 - y3 \* y3;

**double** sx21 = x2 \* x2 - x1 \* x1;

**double** sy21 = y2 \* y2 - y1 \* y1;

**double** f = ((sx13) \* (x12)

+ (sy13) \* (x12)

+ (sx21) \* (x13)

+ (sy21) \* (x13))

/ (2 \* ((y31) \* (x12) - (y21) \* (x13)));

**double** g = ((sx13) \* (y12)

+ (sy13) \* (y12)

+ (sx21) \* (y13)

+ (sy21) \* (y13))

/ (2 \* ((x31) \* (y12) - (x21) \* (y13)));

**double** c = -x1 \* x1 - y1 \* y1 - 2 \* g \* x1 - 2 \* f \* y1;

// eqn of circle be x^2 + y^2 + 2\*g\*x + 2\*f\*y + c = 0

// where centre is (h = -g, k = -f) and radius r

// as r^2 = h^2 + k^2 - c

**double** h = -g;

**double** k = -f;

**double** sqr\_of\_r = h \* h + k \* k - c;

// r is the radius

R = Math.*sqrt*(sqr\_of\_r);

}

**return** R;

}

**public** Point2D getCenter(Point2D p, Point2D q, Point2D r)

{

**double** x1 = p.getX();

**double** y1 = p.getY();

**double** x2 = q.getX();

**double** y2 = q.getY();

**double** x3;

**double** y3;

**double** R = 0;

**if**(r == **null**)

{

**double** centerX = (x1 + x2) / 2;

**double** centerY = (y1 + y2) / 2;

**return** **new** Point2D(centerX, centerY);

}

**if** ((!p.equals(q)) && (!q.equals(r)) && (!p.equals(r)))

{

//----------------------

x3 = r.getX();

y3 = r.getY();

//----------------------

**double** x12 = x1 - x2;

**double** x13 = x1 - x3;

**double** y12 = y1 - y2;

**double** y13 = y1 - y3;

**double** y31 = y3 - y1;

**double** y21 = y2 - y1;

**double** x31 = x3 - x1;

**double** x21 = x2 - x1;

// x1^2 - x3^2

**double** sx13 = x1 \* x1 - x3 \* x3;

// y1^2 - y3^2

**double** sy13 = y1 \* y1 - y3 \* y3;

**double** sx21 = x2 \* x2 - x1 \* x1;

**double** sy21 = y2 \* y2 - y1 \* y1;

**double** f = ((sx13) \* (x12)

+ (sy13) \* (x12)

+ (sx21) \* (x13)

+ (sy21) \* (x13))

/ (2 \* ((y31) \* (x12) - (y21) \* (x13)));

**double** g = ((sx13) \* (y12)

+ (sy13) \* (y12)

+ (sx21) \* (y13)

+ (sy21) \* (y13))

/ (2 \* ((x31) \* (y12) - (x21) \* (y13)));

**double** c = -x1 \* x1 - y1 \* y1 - 2 \* g \* x1 - 2 \* f \* y1;

// eqn of circle be x^2 + y^2 + 2\*g\*x + 2\*f\*y + c = 0

// where centre is (h = -g, k = -f) and radius r

// as r^2 = h^2 + k^2 - c

**double** h = -g;

**double** k = -f;

**double** sqr\_of\_r = h \* h + k \* k - c;

// r is the radius

R = Math.*sqrt*(sqr\_of\_r);

**return** **new** Point2D(h,k);

}

**return** **null**;

}

**private** **double** getAngle(Point2D p, Point2D q, Point2D r)

{

**double** a = (q.getX() - p.getX()) \* (q.getX() - p.getX()) + (q.getY() - p.getY()) \* (q.getY() - p.getY());

**double** b = (q.getX() - r.getX()) \* (q.getX() - r.getX()) + (q.getY() - r.getY()) \* (q.getY() - r.getY());

**double** c = (r.getX() - p.getX()) \* (r.getX() - p.getX()) + (r.getY() - p.getY()) \* (r.getY() - p.getY());

**return** Math.*acos*((a + b - c) / Math.*sqrt*(4 \* a \* b));

}

//for nearest voronoi

**private** Point2D Up1(Point2D p)

{

Point2D nextP = neighbours.get(p).get(1); //O(1)

**double** mx = (p.getX() + nextP.getX()) / 2.0;

**double** my = (p.getY() + nextP.getY()) / 2.0;

**double** dx = nextP.getY() - p.getY();

**double** dy = p.getX() - nextP.getX();

**double** ux = mx + dx;

**double** uy = my + dy;

**return** **new** Point2D(ux, uy);

}

//for farthest voronoi

**private** Point2D Up2(Point2D p)

{

Point2D nextP = neighbours.get(p).get(1); //O(1)

**double** mx = (p.getX() + nextP.getX()) / 2.0;

**double** my = (p.getY() + nextP.getY()) / 2.0;

**double** dx = nextP.getY() - p.getY();

**double** dy = p.getX() - nextP.getX();

**double** ux = mx - dx;

**double** uy = my - dy;

**return** **new** Point2D(ux, uy);

}

**public** **boolean** isPointInCircle(Point2D point, Point2D center, **double** radius)

{

**double** distance = point.distance(center);

**return** distance < radius;

}

**public** **void** updateUp(Point2D p, Point2D c)

{

u\_p.replace(p, c);

}

//-----------O(n)----------------

**public** **void** addAllUpToK()

{

**int** size = getConvexPointsSize();

u\_p = **new** HashMap<Point2D, Point2D>();

**if**(mode == 1) //nearest voronoi

{

**for** (**int** i = 0; i < size; i++)

{

Point2D curr = convexPoints.get(i); //O(1)

Point2D Ucurr = Up1(curr);

u\_p.put(curr, Ucurr);

K.add(Ucurr);

}

}**else** //mode==2 farthest voronoi

{

**for** (**int** i = 0; i < size; i++)

{

Point2D curr = convexPoints.get(i); //O(1)

Point2D Ucurr = Up2(curr);

u\_p.put(curr, Ucurr);

K.add(Ucurr);

}

}

}

**public** **void** addCtoK(Point2D c)

{

K.add(c);

}

**public** **void** addCandUtoE(Point2D c, Point2D up)

{

ArrayList<Point2D> inner = **new** ArrayList<Point2D>();

inner.add(c);

inner.add(up);

E.add(inner);

}

**public** Point2D getUp(Point2D p)

{

**return** u\_p.get(p);

}

**public** **void** makeKandE()

{

K = **new** ArrayList<Point2D>();

E = **new** ArrayList<ArrayList<Point2D>>();

}

**public** **void** moveKandE()

{

allK.add(K);

allE.add(E);

}

**public** **void** initializeKandE()

{

allK = **new** ArrayList<ArrayList<Point2D>>();

allE = **new** ArrayList<ArrayList<ArrayList<Point2D>>>();

}

}

//--------------------------------------------------------------------------------------------------------------

### GrahamScan/Model

Η κλάση ‘GrahamScan’ είναι υπεύθυνη για τον υπολογισμό των σημείων που ορίζουν το Κυρτό Πολύγωνο σε O(nlogn) χρόνο. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

package Model;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import javafx.geometry.Point2D;

public class GrahamScan

{

private static Point2D start;

private ArrayList<Point2D> allPoints;

private ArrayList<Point2D> convexHullPoints;

private DataBase dataBase;

public GrahamScan()

{

dataBase = DataBase.getInstance();

}

public void initialize()

{

allPoints = new ArrayList<Point2D>(dataBase.getAllPoints());

convexHullPoints = new ArrayList<Point2D>(computeGrahamScan(allPoints));

convexHullPoints.remove(convexHullPoints.size()-1);

dataBase.setConvexPoints(convexHullPoints);

}

//----------------------------------O(nlogn)--------------------------------------

private static List<Point2D> computeGrahamScan(List<Point2D> points) {

// Find the point with the lowest y-coordinate (or the leftmost point in case of a tie)

start = points.get(0);

for (Point2D point : points) {

if (point.getY() < start.getY() || (point.getY() == start.getY() && point.getX() < start.getX())) {

start = point;

}

}

// Sort the points by polar angle with respect to the start point

List<Point2D> sortedPoints = new ArrayList<>(points);

sortedPoints.remove(start);

Collections.sort(sortedPoints, (a, b) -> {

double angleA = Math.atan2(a.getY() - start.getY(), a.getX() - start.getX());

double angleB = Math.atan2(b.getY() - start.getY(), b.getX() - start.getX());

if (angleA < angleB) {

return -1;

} else if (angleA > angleB) {

return 1;

} else {

return Double.compare(a.distance(start), b.distance(start));

}

});

// Perform the Graham's scan

List<Point2D> hull = new ArrayList<>();

hull.add(start);

for (Point2D point : sortedPoints) {

while (hull.size() >= 2) {

Point2D p2 = hull.get(hull.size() - 1);

Point2D p1 = hull.get(hull.size() - 2);

if (crossProduct(p1, p2, point) < 0)

{

hull.remove(hull.size() - 1);

} else

{

break;

}

}

hull.add(point);

}

hull.add(start);

return hull;

}

private static double crossProduct(Point2D p, Point2D q, Point2D r) {

double y1 = q.getY() - p.getY();

double y2 = r.getY() - p.getY();

double x1 = q.getX() - p.getX();

double x2 = r.getX() - p.getX();

return x1 \* y2 - x2 \* y1;

}

}

//--------------------------------------------------------------------------------------------------------------

### NeighborVoronoiDiagram/Model

Η κλάση ‘NeighborVoronoiDiagram’ είναι υπεύθυνη για τον υπολογισμό των σημείων που ορίζουν τα διαγράμματα Voronoi(κοντινότερου και μακρύτερου) γείτονα σε O(nlogn) χρόνο. Ουσιαστικά, η ίδια κλάση χρησιμοποιείται δύο φορές ξεχωριστά ώστε την πρώτη φορά να υπολογίσει το Διάγραμμα Voronoi του κοντινότερου γείτονα και την δεύτερη φορά για να υπολογίσει το Διάγραμμα Voronoi του μακρύτερου γείτονα. Ο κάθε αλγόριθμος χρειάζεται O(nlogn) χρόνο. Οπότε και δύο μαζί χρειάζονται O(nlogn). Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Model;

**import** javafx.geometry.Point2D;

**public** **class** NeighborVoronoiDiagram

{

**private** DataBase dataBase;

**public** NeighborVoronoiDiagram()

{

dataBase = DataBase.*getInstance*();

}

**public** **void** computeVoronoiDiagram()

{

System.***out***.println("I am the second algorithm!");

**int** n = dataBase.getConvexPointsSize();

//I have to make (K,E) ArrayLits for each K, E

dataBase.makeKandE();

//for all p in S add u(p) to K;

dataBase.addAllUpToK();

Point2D before\_p;

Point2D p;

Point2D next\_p;

Point2D c;

**if**(n > 2)

{

System.***out***.println("Algorithm 2 started with = " + n);

**do**

{

//find p maximizing radius and angle

p = dataBase.findMaxP();

before\_p = dataBase.getPrev();

next\_p = dataBase.getNext();

c = dataBase.getCenter(before\_p, p, next\_p);

dataBase.addCtoK(c);

dataBase.addCandUtoE(c, dataBase.getUp(p));

dataBase.addCandUtoE(c, dataBase.getUp(before\_p));

dataBase.updateUp(before\_p, c);

dataBase.deleteMaxP();

n = n-1;

}**while**(n != 2);

dataBase.addCandUtoE(c, dataBase.getUp(next\_p));

}**else**

{

System.***out***.println("Algorithm 2 started with = " + n);

**if**(n == 2) //S={p1, p2}

{

Point2D p1 = dataBase.getConvexPoints().get(0);

Point2D p2 = dataBase.getConvexPoints().get(1);

Point2D u\_p1 = dataBase.getUp(p1);

Point2D u\_p2 = dataBase.getUp(p2);

//add (u(p1), u(p2)) to E;

dataBase.addCandUtoE(u\_p1, u\_p2);

}

}

dataBase.moveKandE();

System.***out***.println("Algorithm 2 finished!");

}

}

//--------------------------------------------------------------------------------------------------------------

### SmallestEnclosingCircle/Model

Η κλάση ‘SmallestEnclosingCircle’ είναι υπεύθυνη για τον υπολογισμό των σημείων που ορίζουν τα σημεία του Ελάχιστου Περικλείοντος Κύκλου σε O(nlogn) χρόνο. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Model;

**import** java.lang.Math;

**public** **class** SmallestEnclosingCirlce

{

**private** DataBase dataBase;

**public** SmallestEnclosingCirlce()

{

dataBase = DataBase.*getInstance*();

}

**public** **void** computeSmallestEnclosingCircle()

{

**boolean** finish;

**double** maxAngle;

**double** myPi2 = Math.***PI***/2;

**if**(dataBase.getHashCirclePointsSize() != 1)

{

finish = **false**;

**do**

{

maxAngle = dataBase.findMaxP().getX();

**if**(maxAngle>myPi2)

{

dataBase.deleteMaxP();

}**else**

{

dataBase.moveToCircleArray();

System.***out***.println("The algorithm 1 finished = "+dataBase.getHashCirclePointsSize());

finish = **true**;

}

}**while**(!finish);

}**else**

{

System.***out***.println("Only 1 point");

}

}

}

//--------------------------------------------------------------------------------------------------------------

### TextHandler/Model

Η κλάση ‘TextHandler’ είναι υπεύθυνη για την αποθήκευση των κειμένων για όλα τα κουμπιά και κείμενα του προγράμματος καθώς επίσης και την εναλλαγή τους μεταξύ Ελληνικών-Αγγλικών. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

**package** Model;

**public** **class** TextHandler

{

**private** **static** TextHandler *instance*;

//-----------parallel---------------

**private** String changeLanguageButtonText = "EN";

**private** String paperTitleText = "A SIMPLE ALGORITHM FOR COMPUTING THE SMALLEST ENCLOSING CIRCLE\n\t";

**private** String loadFileButtonText = "Load Points From File";

**private** String putPointsButtonText = "Create Points";

**private** String exitButtonText = "Exit";

//----------notParallel----------------

**private** String fileChooserTitleText = "Choose the file who contains the points";

**private** String notImpementPopupTitle = "Put points";

**private** String notImpementPopupHeaderText = "Under construction!";

**private** String notImpementContentText = "The manual addition of points has not been implemented yet!\\nIn the future the user will be able to add points manually and modify them without having to load them from an external file.";

**private** String infoWindowTitle = "Information + Instructions for use";

**private** String infoContent = "ABOUT\n"

+ "This thesis was developed in the context of the paper by Sven Skyum 1991 entitled \"A SIMPLE ALGORITHM FOR COMPUTING THE SMALLEST ENCLOSING CIRCLE\" and concerns the implementation and visualization of an efficient complexity algorithm O(nlogn) and computes the least pericycle and with small changes and maintaining the same complexity is able to compute both the Voronoi diagrams of the nearest and the farthest neighbor\r\n\n"

+ "IMPLEMENTATION INSTRUCTIONS:\r\n"

+ "1) You can change the language from Greek to English and vice versa in real time by pressing the button on the top right with the corresponding mark\r\n"

+ "2) By pressing the 'Exit' button the application terminates\r\n"

+ "3) Pressing the 'Load Points' button will display a window asking you to load a file with points x,y from the supported application types (csv, txt, excel)\r\n"

+ "4) Finally, the three charts with their corresponding graphs will be displayed.\r\n"

+ "5) You can do the following actions on the graphs:\r\n"

+ " - alt+leftclick: drag and drop\r\n"

+ " - rightclick: export to file\r\n"

+ " - leftclick: choose a point \r\n"

+ " - scroll: zoom in or zoom out\r\n"

+ "\*\* The 'Create Points' button has not been implemented yet and will be implemented at some point in the future\r\n"

+ "\*\*\* Leftclicking on the title will open the browser in the thesis paper\r\n\n"

+ "IMAGES FROM THE APP\n";

**private** TextHandler()

{

//TextClass();

}

//Get the only object available

**public** **static** TextHandler getInstance()

{

**if** (*instance* == **null**)

{

*instance* = **new** TextHandler();

}

**return** *instance*;

}

**public** **void** setLanguage(String language)

{

changeLanguageButtonText = language;

**if**(language.equals("EN"))

{

//-----MainWidnow--------------

paperTitleText = "A SIMPLE ALGORITHM FOR COMPUTING THE SMALLEST ENCLOSING CIRCLE\n\t";

loadFileButtonText = "Load Points From File";

putPointsButtonText = "Create Points";

exitButtonText = "Exit";

//-----FileChooserWindow--------------

fileChooserTitleText = "Choose the file who contains the points";

//PutPointsButtonHandler popup

notImpementPopupTitle = "Put points";

notImpementPopupHeaderText = "Under construction!";

notImpementContentText = "The manual addition of points has not been implemented yet!\nIn the future the user will be able to add points manually and modify them without having to load them from an external file.";

//Info button

infoWindowTitle = "Information + Instructions for use";

infoContent = "ABOUT\n"

+ "This thesis was developed in the context of the paper by Sven Skyum 1991 entitled \"A SIMPLE ALGORITHM FOR COMPUTING THE SMALLEST ENCLOSING CIRCLE\" and concerns the implementation and visualization of an efficient complexity algorithm O(nlogn) and computes the least pericycle and with small changes and maintaining the same complexity is able to compute both the Voronoi diagrams of the nearest and the farthest neighbor\r\n\n"

+ "IMPLEMENTATION INSTRUCTIONS:\r\n"

+ "1) You can change the language from Greek to English and vice versa in real time by pressing the button on the top right with the corresponding mark\r\n"

+ "2) By pressing the 'Exit' button the application terminates\r\n"

+ "3) Pressing the 'Load Points' button will display a window asking you to load a file with points x,y from the supported application types (csv, txt, excel)\r\n"

+ "4) Finally, the three charts with their corresponding graphs will be displayed.\r\n"

+ "5) You can do the following actions on the graphs:\r\n"

+ " - alt+leftclick: drag and drop\r\n"

+ " - rightclick: export to file\r\n"

+ " - leftclick: choose a point \r\n"

+ " - scroll: zoom in or zoom out\r\n"

+ "\*\* The 'Create Points' button has not been implemented yet and will be implemented at some point in the future\r\n"

+ "\*\*\* Leftclicking on the title will open the browser in the thesis paper\r\n\n"

+ "IMAGES FROM THE APP\n";

}**else** //GR

{

//-----MainWidnow--------------

paperTitleText = "ΕΝΑΣ ΑΠΛΟΣ ΑΛΓΟΡΙΘΜΟΣ ΓΙΑ ΤΟΝ ΥΠΟΛΟΓΙΣΜΟ ΤΟΥ ΕΛΑΧΙΣΤΟΥ ΠΕΡΙΚΛΕΙΟΝΤΑ ΚΥΚΛΟΥ";

loadFileButtonText = "Φόρτωση Σημείων από Αρχείο";

putPointsButtonText = "Δημιουργία σημείων";

exitButtonText = "Έξοδος";

//-----FileChooserWindow--------------

fileChooserTitleText = "Επιλέξτε το αρχείο που περιέχει τα σημεία";

//PutPointsButtonHandler popup

notImpementPopupTitle = "Προσθήκη Σημείων";

notImpementPopupHeaderText = "Υπό κατασκευή!";

notImpementContentText = "Η χειροκίνητη προσθήκη σημείων δεν έχει υλοποιηθεί ακόμα!\nΜελλοντικά ο χρήστης θα μπορεί να προσθέσει σημεία με το χέρι του και να τα τροποποιεί χωρίς να χρειάζεται η φόρτωση τους από εξωτερικό αρχείο.";

//Info button

infoWindowTitle = "Πληροφορίες + Οδηγίες χρήσης";

infoContent = "ΣΧΕΤΙΚΑ\n"

+ "Αυτή η διπλωματική εργασία αναπτύχθηκε στα πλαίσια του Paper από τον Sven Skyum 1991 με τίτλο \"A SIMPLE ALGORITHM FOR COMPUTING THE SMALLEST ENCLOSING CIRCLE\" και αφορά την υλοποίηση και οπτικοποίηση ενός αποδοτικού αλγορίθμου πολυπλοκότητας O(nlogn) και υπολογίζει τον ελάχιστο περικλέιοντα κύκλο και με μικρές αλλαγές και διατηρώντας την ίδια πολυπλοκότητα είναι ικανός να υπολογίσει και τα διαγράμματα Voronoi του κοντινότερου και του μακρύτερου γείτονα\r\n\n"

+ "ΟΔΗΓΙΕΣ ΕΦΑΡΜΟΓΗΣ:\r\n"

+ "1) Μπορείτε να αλλάξετε την γλώσσα από ελληνικά σε αγγλικά και αντίστροφα σε πραγματικό χρόνο πατώντας το κουμπί πάνω δεξιά με την αντίστοιχει σήμανση\r\n"

+ "2) Πατώντας το κουμπί 'Έξοδος' η εφαρμογή τερματίζει\r\n"

+ "3) Πατώντας το κουμπί 'Φόρτωση Σημείων' θα σας εμφανιστεί ένα παράθυρο στο οποίο θα σας ζητηθεί να φορτώσετε ένα αρχείο με σημεία x,y από τους υποστηριζόμενους τύπους της εφαρμογής(csv, txt, excel)\r\n"

+ "4) Τέλος, εμφανίζονται τα τρία διαγράμματα με τα αντίστοιχα γραφήματα.\r\n"

+ "5) Μπορείτε να κάνετε τις εξής ενέργειες στα γραφήματα:\r\n"

+ " - alt+αριστερό-κλικ: μετακίνηση\r\n"

+ " - δεξί-κλικ: export to file\r\n"

+ " - leftclick: choose a point \r\n"

+ " - scroll: zoom in or zoom out\r\n"

+ "\*\* Το κουμπί 'Δημιουργία Σημείων' δεν έχει υλοποιηθεί ακόμα και θα υλοποιηθεί κάποια στιγμή στο μέλλον\r\n"

+ "\*\*\* Κάνοντας αριστερό-κλικ στον τίτλο θα ανοίξει ο browser στο paper της δηπλωματικής εργασίας\n\n"

+ "ΕΙΚΟΝΕΣ ΑΠΟ ΤΗΝ ΕΦΑΡΜΟΓΗ\n";

}

}

**public** String getChangeLanguageButtonText()

{

**return** changeLanguageButtonText;

}

**public** String getPaperTitleText()

{

**return** paperTitleText;

}

**public** String getLoadFileButtonText()

{

**return** loadFileButtonText;

}

**public** String getPutPointsButtonText()

{

**return** putPointsButtonText;

}

**public** String getExitButtonText()

{

**return** exitButtonText;

}

**public** String getFileChooserTitleText()

{

**return** fileChooserTitleText;

}

**public** String getNotImpementPopupTitle()

{

**return** notImpementPopupTitle;

}

**public** String getNotImpementPopupHeaderText()

{

**return** notImpementPopupHeaderText;

}

**public** String getNotImpementContentText()

{

**return** notImpementContentText;

}

**public** String getInfoWindowTitle()

{

**return** infoWindowTitle;

}

**public** String getInfoContent()

{

**return** infoContent;

}

}

//--------------------------------------------------------------------------------------------------------------

### CSVLoader/Model\_Loaders

Η κλάση ‘CSVLoader’ είναι υπεύθυνη για την ανάγνωση και αποθήκευση των σημείων από .csv αρχεία. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

package Model\_Loaders;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import com.opencsv.\*;

import com.opencsv.exceptions.CsvException;

import javafx.geometry.Point2D;

public class CSVLoader implements FileLoader

{

private String csvPath;

private List<String[]> stringCSVData;

private ArrayList<Point2D> points2DCSVData;

public CSVLoader(String csvPath) throws IOException, CsvException

{

this.csvPath=csvPath;

stringCSVData = new ArrayList<String[]>();

points2DCSVData = new ArrayList<Point2D>();

System.out.println("I am CSV LOADER");

readValues();

convertStringToPoint2D();

}

private void readValues() throws IOException, CsvException

{

CSVParser csvParser = new CSVParserBuilder().withSeparator(',').build();

CSVReader csvReader = new CSVReaderBuilder(new FileReader(csvPath))

// .withCSVParser(csvParser)

.withSkipLines(0).build();

stringCSVData.addAll(csvReader.readAll());

}

public void convertStringToPoint2D()

{

//Convert data from String to double

double a;

double b;

String temp1;

String temp2;

for(int i=0; i<stringCSVData.size(); i++)

{

temp1 = stringCSVData.get(i)[0].replace(',', '.');

temp2 = stringCSVData.get(i)[1].replace(',', '.');

a=Double.parseDouble(temp1);

b=Double.parseDouble(temp2);

points2DCSVData.add(new Point2D(a, b));

}

}

public ArrayList<Point2D> get2Dvalues()

{

return points2DCSVData;

}

}

//--------------------------------------------------------------------------------------------------------------

### ExcelLoader/Model\_Loader

Η κλάση ‘ExcelLoader’ είναι υπεύθυνη για την ανάγνωση και αποθήκευση των σημείων από .xlsx ή .xls αρχεία. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

package Model\_Loaders;

import org.apache.poi.ss.usermodel.\*;

import java.io.File;

import java.io.FileInputStream;

import java.util.ArrayList;

import org.apache.poi.ss.usermodel.Cell;

import org.apache.poi.ss.usermodel.Row;

import javafx.geometry.Point2D;

public class ExcelLoader implements FileLoader

{

private String path;

private ArrayList<Point2D> points2DXLSXData;

public ExcelLoader(String path)

{

this.path = path;

points2DXLSXData = new ArrayList<Point2D>();

System.out.println("I am EXCEL LOADER");

readValues();

}

private void readValues()

{

try {

// Create a FileInputStream object to read the Excel file

FileInputStream fis = new FileInputStream(new File(path));

// Create a Workbook object to read the Excel file

Workbook workbook = WorkbookFactory.create(fis);

// Get the first sheet from the workbook

Sheet sheet = workbook.getSheetAt(0);

// Iterate through the rows of the sheet

for (Row row : sheet)

{

convertDoubleToPoint2D(row.getCell(0), row.getCell(1));

}

// Close the FileInputStream object

fis.close();

} catch (Exception e) {

e.printStackTrace();

}

}

public void convertDoubleToPoint2D(Cell column1, Cell column2)

{

double a = column1.getNumericCellValue();

double b = column2.getNumericCellValue();

points2DXLSXData.add(new Point2D(a, b));

}

@Override

public ArrayList<Point2D> get2Dvalues()

{

return points2DXLSXData;

}

}

//--------------------------------------------------------------------------------------------------------------

### FileLoader/Model\_Loader

Το Interface ‘FileLoader’ ορίζει ότι όλες οι κλάσεις που διαβάζουν διάφορους τύπους αρχείων θα πρέπει να υλοποιούν την μέθοδο ‘get2DValues()’ η οποία αποθηκεύει όλα τα σημεία σε μια ArrayList<Point2D> και τα επιστρέφει.

//--------------------------------------------------------------------------------------------------------------

**package** Model\_Loaders;

**import** java.util.ArrayList;

**import** javafx.geometry.Point2D;

**public** **interface** FileLoader

{

**public** ArrayList<Point2D> get2Dvalues();

}

//--------------------------------------------------------------------------------------------------------------

### FileLoaderFactory/Model\_Loaders

Η Factory κλάση ‘ExcelLoaderFactory’ είναι υπεύθυνη για την δημιουργία του αντίστοιχου αντικειμένου με βάση την κατάληξη του αρχείου. Πιο συγκεκριμένα, αν το αρχείο έχει κατάληξη .csv τότε θα δημιουργήσει ένα αντικείμενο CSVLoader και θα εκκινήσει την διαδικασία ανάγνωσης η κλάση αυτή. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της

//--------------------------------------------------------------------------------------------------------------

**package** Model\_Loaders;

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** com.opencsv.exceptions.CsvException;

**import** Model.DataBase;

**import** javafx.geometry.Point2D;

**public** **class** FileLoaderFactory

{

**private** String path;

**private** CSVLoader csvLoader;

**private** TXTLoader txtLoader;

**private** ExcelLoader xlsxLoader;

**private** ArrayList<Point2D> allPoints;

**private** DataBase dataBase;

**public** FileLoaderFactory(String path)

{

**this**.path = path;

dataBase = DataBase.*getInstance*();

}

**public** ArrayList<Point2D> getAllPoints()

{

**try** {

**if**(path.contains(".csv"))

{

csvLoader = **new** CSVLoader(path);

allPoints = **new** ArrayList<Point2D>(csvLoader.get2Dvalues());

}**else** **if**(path.contains(".txt"))

{

txtLoader = **new** TXTLoader(path);

allPoints = **new** ArrayList<Point2D>(txtLoader.get2Dvalues());

}**else** **if**(path.contains(".xlsx") || path.contains(".xls"))

{

xlsxLoader = **new** ExcelLoader(path);

allPoints = **new** ArrayList<Point2D>(xlsxLoader.get2Dvalues());

}**else**

{

}

} **catch** (IOException | CsvException e)

{

e.printStackTrace();

}

dataBase.setAllPoints(allPoints);

**return** allPoints;

}

}

//--------------------------------------------------------------------------------------------------------------

### TXTLoader/Model\_Loader

Η κλάση ‘TXTLoader’ είναι υπεύθυνη για την ανάγνωση και αποθήκευση των σημείων από .txt αρχεία. Το όνομα της κάθε μεθόδου είναι συνώνυμο με την λειτουργία της.

//--------------------------------------------------------------------------------------------------------------

package Model\_Loaders;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Scanner;

import com.opencsv.exceptions.CsvException;

import javafx.geometry.Point2D;

public class TXTLoader implements FileLoader

{

private String path;

private ArrayList<Point2D> points2DTXTData;

public TXTLoader(String path) throws IOException, CsvException

{

this.path = path;

points2DTXTData = new ArrayList<Point2D>();

System.out.println("I am TXT LOADER");

readValues();

}

private void readValues() throws IOException, CsvException

{

File file = new File(path);

Scanner scanner = new Scanner(file);

String line;

String[] values ;

while (scanner.hasNextLine())

{

line = scanner.nextLine();

values = line.split("\\s+"); // assuming the values are separated by whitespace

convertStringToPoint2D(values[0], values[1]);

}

scanner.close();

}

public void convertStringToPoint2D(String x, String y)

{

String temp1 = x.replace(',', '.');

String temp2 = y.replace(',', '.');

double a = Double.parseDouble(temp1);

double b = Double.parseDouble(temp2);

points2DTXTData.add(new Point2D(a, b));

}

@Override

public ArrayList<Point2D> get2Dvalues()

{

return points2DTXTData;

}

}

//--------------------------------------------------------------------------------------------------------------

## Παραδείγματα Εκτέλεσης του Προγράμματος

# Επίλογος

# Βιβλιογραφία

1. Graham Scan Paper: <https://www.sciencedirect.com/science/article/pii/0020019072900452?via%3Dihub>
2. Graham Scan Images: <https://en.wikipedia.org/wiki/Graham_scan#cite_note-g72-1>