



ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

**Πρόγραμμα Σπουδών
Τμήματος Μηχανικών Πληροφορικής Τ.Ε. Λάρισας**

**Δημιουργία εφαρμογών
βασισμένων σε συστατικά
λογισμικού με αναδόμηση
εφαρμογών Java σε ενότητες**

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Ζάχος Ηλίας (ΑΜ: 4415044)

Επιβλέπων: Κακαρόντζας Γεώργιος, Επίκουρος Καθηγητής

ΛΑΡΙΣΑ 2019

«Εγώ ο Ζάχος Ηλίας, δηλώνω υπεύθυνα ότι η παρούσα Πτυχιακή Εργασία με τίτλο ‘Δημιουργία εφαρμογών βασισμένων σε συστατικά λογισμικού με αναδόμηση εφαρμογών Java σε ενότητες’ είναι δική μου και βεβαιώνω ότι:

- Σε όσες περιπτώσεις έχω συμβουλευτεί δημοσιευμένη εργασία τρίτων, αυτό επισημαίνεται με σχετική αναφορά στα επίμαχα σημεία.*
- Σε όσες περιπτώσεις μεταφέρω λόγια τρίτων, αυτό επισημαίνεται με σχετική αναφορά στα επίμαχα σημεία. Με εξαίρεση τέτοιες περιπτώσεις, το υπόλοιπο κείμενο της πτυχιακής αποτελεί δική μου δουλειά.*
- Αναφέρω ρητά όλες τις πηγές βοήθειας που χρησιμοποίησα.*
- Σε περιπτώσεις που τμήματα της παρούσας πτυχιακής έγιναν από κοινού με τρίτους, αναφέρω ρητά ποια είναι η δική μου συνεισφορά και ποια των τρίτων.*
- Γνωρίζω πως η λογοκλοπή αποτελεί σοβαρότατο παράπτωμα και είμαι ενήμερος για την επέλευση των νόμιμων συνεπειών»*

Ζάχος Ηλίας

Ο φοιτητής εντάχθηκε αυτοδίκαια στο Πανεπιστήμιο Θεσσαλίας, σύμφωνα με την παρ. 1 του άρθρου 6 του Ν.4589/2019 (ΦΕΚ 13/Α'/29.01.2019). Η εκπαιδευτική λειτουργία του ανωτέρου προγράμματος σπουδών συνεχίζεται μεταβατικά σύμφωνα με την παρ. 2 του άρθρου 6 του Ν.4589/2019 (ΦΕΚ 13/Α'/29.01.2019).

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Τόπος:

Ημερομηνία:

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1.
2.
3.

Περίληψη

Η παρακάτω πτυχιακή εργασία αφορά την δημιουργία ενός plugin για το Eclipse που θα αναλύει δομικά τον πηγαίο κώδικα κάποιου project που είναι γραμμένο σε Java. Η εφαρμογή αυτή θα αποτελέσει μια βοήθεια για τον προγραμματιστή που γράφει σε Java και θέλει να ομαδοποιήσει συγγενικά πακέτα σε modules με βάση μια απλοϊκή μετρική ($\frac{\text{αριθμός των κλάσεων που καλούνται από ένα πακέτο}}{\text{αριθμός των κλάσεων που έχει το πακέτο}}$).

Η ανάλυση του κώδικα επιτυγχάνεται με την χρήση της εσωτερικής βιβλιοθήκης «CallHierarchy» του Eclipse και η γραφική παράσταση της ιεραρχικής συσταδοποίησης των πακέτων έγινε χρησιμοποιώντας την βιβλιοθήκη «RCaller».

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την οικογένειά μου, τους στενούς μου φίλους, καθώς και την κοπέλα μου που με παρότρυναν να αναλάβω και να φέρω εις πέρας την παρούσα πτυχιακή εργασία και στάθηκαν δίπλα μου σε όσες δύσκολες στιγμές υπήρξαν. Επιπλέον, θα ήθελα να ευχαριστήσω τους καθηγητές μου που με βοήθησαν να ανεβάσω το γνωστικό μου επίπεδο. Και τέλος, θα ήθελα φυσικά να ευχαριστήσω τον κύριο Κακαρόντζα για τις συμβουλές που μου έδωσε και την καθοδήγηση που έλαβα για την περάτωση αυτής της εργασίας.

Ζάχος Ηλίας

04/09/2019

Περιεχόμενα

ΠΕΡΙΛΗΨΗ	I
ΕΥΧΑΡΙΣΤΙΕΣ.....	III
ΠΕΡΙΕΧΟΜΕΝΑ.....	V
1 ΕΙΣΑΓΩΓΗ	1
2 ΘΕΩΡΗΤΙΚΕΣ ΒΑΣΙΚΕΣ ΑΡΧΕΣ.....	3
2.1 ECLIPSE.....	3
2.1.1 Εισαγωγή Βιβλιοθήκης σε Project	3
2.1.2 Δημιουργία Συντόμευσης plug-in	4
2.2 ΣΥΣΤΑΔΟΠΟΙΗΣΗ	10
2.2.1 Μέθοδοι συσταδοποίησης.....	10
2.2.2 Ιεραρχική συσταδοποίηση	11
3 JAVAMODEL ΚΑΙ AST	15
3.1 JAVAMODEL	16
3.1.1 IPackageFragment.....	16
3.1.2 ICompilationUnit.....	17
3.1.3 IMethod.....	17
3.1.4 IMethodBinding.....	17
3.1.5 ITypeBinding.....	18
3.2 AST.....	19
3.2.1 ASTVisitor.....	19
3.2.2 ASTNode.....	20
3.2.3 ASTParser.....	20
3.2.4 NodeFinder.....	21
4 CALLHIERARCHY	23
4.1 ΚΛΑΣΗ CALLHIERARCHY	23
4.2 ΚΛΑΣΗ METHODWRAPPER	24

5	R.....	25
5.1	RCALLER.....	28
5.2	ΜΕΤΑΦΟΡΑ ΤΩΝ ΔΕΔΟΜΕΝΩΝ	29
5.3	ΤΡΟΠΟΙ ΕΚΤΕΛΕΣΗΣ ΚΩΔΙΚΑ ΣΤΗΝ R.....	32
6	ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ.....	36
6.1	Ο ΠΥΡΗΝΑΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ	36
6.1.1	<i>MethodCA</i>	38
6.1.2	<i>CallerCA</i>	40
6.1.3	<i>MethodDeclarationFinder</i>	41
6.1.4	<i>HierarchyCallHelper</i>	42
6.1.5	<i>GetInfo</i>	44
6.2	ΕΜΦΑΝΙΣΗ ΤΟΥ ΔΙΑΓΡΑΜΜΑΤΟΣ	51
7	ΣΥΜΠΕΡΑΣΜΑΤΑ.....	55
	ΒΙΒΛΙΟΓΡΑΦΙΑ	57

1 Εισαγωγή

Η εργασία χωρίζεται σε δύο στάδια: την δημιουργία ενός plugin για το IDE του Eclipse το οποίο θα αναλύει τον κώδικα κάποιου “ανοιχτού” project και θα εξάγει ως αποτέλεσμα ένα αρχείο με τις συσχετίσεις που υπάρχουν μεταξύ των πακέτων και σε τι βαθμό βρίσκονται αυτές, καθώς και την ανάπτυξη μιας εφαρμογής που θα δέχεται ως είσοδο το αρχείο που εξήγαγε το προηγούμενο πρόγραμμα και θα δημιουργεί ένα διάγραμμα ιεραρχικής ομαδοποίησης, όπου έπειτα θα το εμφανίζει.

Στην υλοποίηση του πρώτου σταδίου χρησιμοποιήθηκε για την εύρεση των συσχετίσεων των κλάσεων μια εσωτερική βιβλιοθήκη του Eclipse, το «CallHierarchy», επίσης χρησιμοποιήθηκαν το AST (Abstract Syntax Tree) και το JavaModel του Eclipse που βοήθησαν στην φόρτωση του project και στην εξαγωγή των βασικών στοιχείων αυτού, παραδείγματος χάριν για τις μεθόδους εξάχθηκε το όνομα, οι παράμετροι, ο τύπος δεδομένων που επιστρέφουν, η κλάση και το πακέτο στα οποία αυτές βρίσκονται, καθώς και το modifier που έχουν. Το project χάρη στο AST δεν χρειάστηκε να εκτελείται για να παρθούν τα στοιχεία και έτσι η ανάλυσή του είναι στατική.

Για την επίτευξη του δεύτερου σταδίου χρησιμοποιήθηκε η βιβλιοθήκη «RCaller», η οποία μας επιτρέπει μέσω της Java να καλέσουμε κώδικα της γλώσσας R. Το R είναι ένα στατιστικό πακέτο και μας βοηθάει να εμφανίσουμε το γράφημα της ιεραρχικής ομαδοποίησης των πακέτων και να συμπεράνουμε ποια πακέτα θα ήταν καλό να συγχωνευτούν και ποια όχι, με βάση κάποια συγκεκριμένη μετρική.

2 Θεωρητικές Βασικές Αρχές

Αυτή η ενότητα θα αποτελέσει την εξήγηση κάποιων βασικών αρχών αυτής της εργασίας. Όπως, τι είναι το Eclipse, πως μπορούμε να χρησιμοποιήσουμε ξένες βιβλιοθήκες, πως βάζουμε προγραμματιστικά συντομεύσεις για plug-ins στο toolbar, καθώς και στο κυρίως menu του Eclipse, τι είναι η συσταδοποίηση και πιο συγκεκριμένα η ιεραρχική.

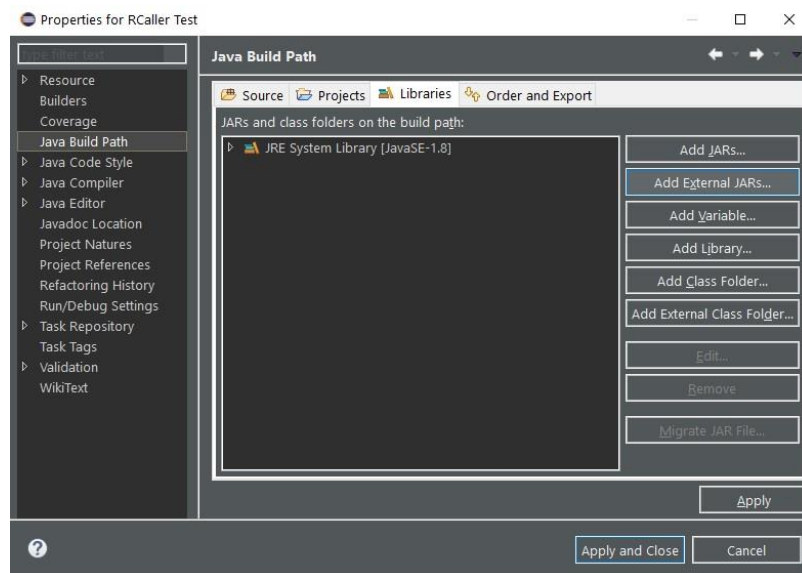
2.1 Eclipse

Το Eclipse είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) που χρησιμοποιείται στον προγραμματισμό ηλεκτρονικών υπολογιστών και περιέχει ένα μεγάλο εύρος από plug-ins για την παραμετροποίηση του περιβάλλοντος. Είναι γραμμένο πιο πολύ σε Java και η κύρια χρήση του είναι η ανάπτυξη λογισμικού σε Java, όμως επιτρέπει και την ανάπτυξη λογισμικού σε άλλες γλώσσες με την χρήση κάποιων plug-ins. Επιπλέον τα περιβάλλοντα ανάπτυξης συμπεριλαμβάνουν και το JDT (Java Development Tools), το οποίο περιέχει κάποιες βιβλιοθήκες, οι οποίες μας είναι χρήσιμες και θα αναφερθούν περισσότερα για αυτές στις επόμενες ενότητες.

2.1.1 Εισαγωγή Βιβλιοθήκης σε Project

Όταν γίνεται ανάπτυξη λογισμικού, ορισμένες φορές θέλουμε να χρησιμοποιήσουμε κάποια συγκεκριμένα χαρακτηριστικά, τα οποία όμως έχουν ήδη γραφτεί από άλλους προγραμματιστές. Η χρήση τέτοιων βιβλιοθηκών αποτελεί μια άμεση λύση για την επίτευξη του στόχου μας και μπορούμε να παραλείψουμε την χρονοβόρα διαδικασία της δημιουργίας μιας νέας βιβλιοθήκης. Έτσι, μπορούμε να επικεντρωθούμε πιο πολύ στον λειτουργικό κώδικα της εφαρμογής. Στην εργασία αυτή, χρησιμοποιήθηκε η βιβλιοθήκη RCaller για να καλέσουμε κώδικα της R από την Java. Έτσι, επειδή υπήρχε ήδη κάποια βιβλιοθήκη που πετύχαινε αυτόν τον σκοπό δεν χρειαζόταν να αναπτυχθεί κάποια καινούρια. Μπορούμε είτε να χρησιμοποιήσουμε το maven, το οποίο μας επιτρέπει να εισάγουμε βιβλιοθήκες, είτε κατευθείαν μέσω του IDE που είναι και πιο εύκολος τρόπος, αλλά θα πρέπει να έχουμε κατεβάσει την βιβλιοθήκη σε μορφή .jar. Παρακάτω θα παρουσιαστεί ένας τρόπος που μπορούμε μέσω του Eclipse να εισάγουμε βιβλιοθήκες σε κάποιο project μας.

Ως παράδειγμα, θα τεθεί η εισαγωγή της βιβλιοθήκης RCaller στο project που θα έχουμε τον κώδικα για την εμφάνιση της ιεραρχικής συσταδοποίησης των πακέτων. Εφόσον, έχουμε δημιουργήσει το project, επιλέγουμε το project από τον project explorer, πατάμε alt + enter (συντόμευση για να ανοίξουμε τα properties του project) και μας εμφανίζει το εξής παράθυρο.



Όπως φαίνεται έχουμε επιλέξει την ιδιότητα Java Build Path από το αριστερό μενού και μετά επιλέξαμε τα Libraries. Έπειτα, θα πατήσουμε “Add External JARs...”, θα μας ανοίξει ένα dialog, στο οποίο θα επιλέξουμε το .jar αρχείο της βιβλιοθήκης που θέλουμε να χρησιμοποιήσουμε. Μετά, θα πατήσουμε “Apply and Close”.

Έτσι τώρα μπορούμε να χρησιμοποιήσουμε τα πακέτα αυτής της βιβλιοθήκης, γράφοντας απλά τα κατάλληλα import στον κώδικά μας. Παραδείγματος χάριν, για να χρησιμοποιήσουμε τις κλάσεις της βιβλιοθήκης RCaller, θα γράψουμε τα εξής imports.

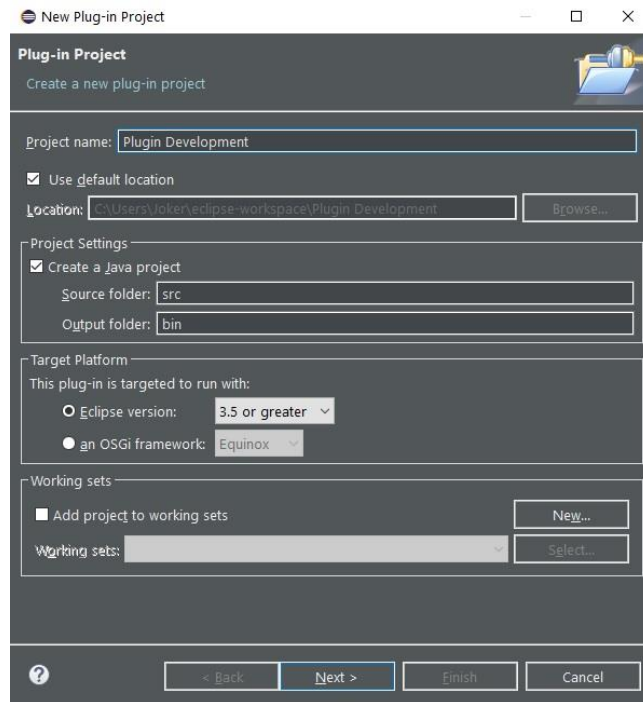
```
import com.github.rcaller.rstuff.RCaller;
import com.github.rcaller.rstuff.RCode;
```

2.1.2 Δημιουργία Συντόμευσης plug-in

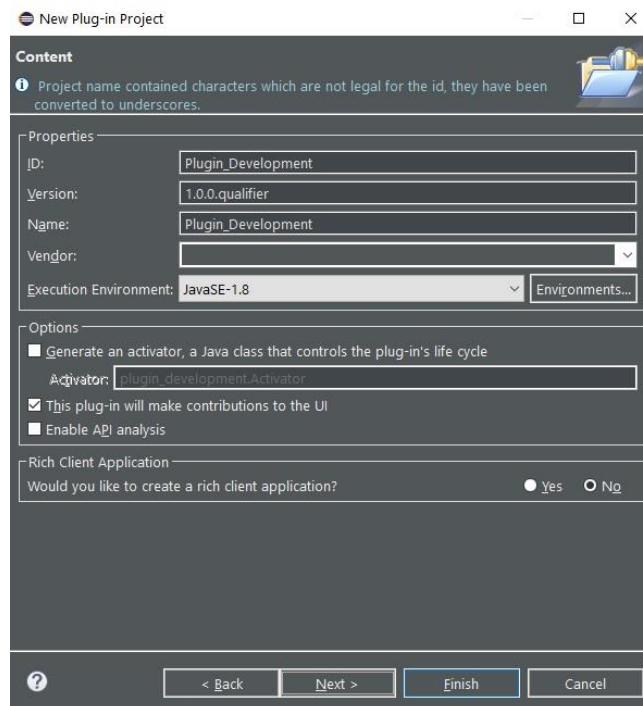
Για να εκτελεστεί ο κώδικας που γράψαμε στο plug-in θα πρέπει να κληθεί από το eclipse. Ένας τρόπος για να γίνει αυτό είναι να φτιάξουμε κάποιο command για την εκτέλεση αυτού του κώδικα. Για να απλουστευτεί η όλη διαδικασία θα χρησιμοποιηθεί το Plug-in Project template του eclipse.

Έτσι θα δημιουργήσουμε ένα καινούριο Plug-in Project, πατώντας στο menu του eclipse: File > New > Plug-in Project. Όταν το πατήσουμε θα ανοίξει ένα παράθυρο, το οποίο θα

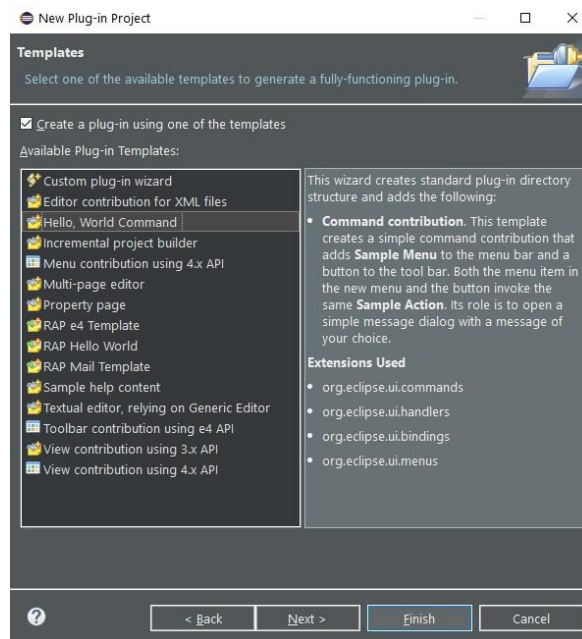
μας ζητάει να εισάγουμε κάποια στοιχεία για το project που θα δημιουργηθεί, όπως φαίνονται στην παρακάτω εικόνα.



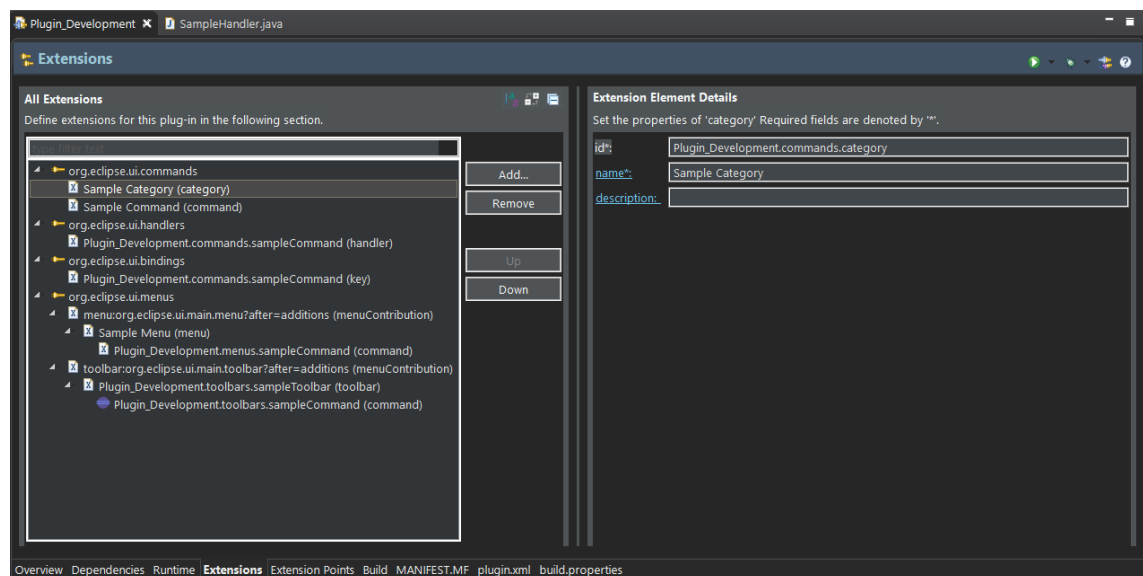
Εφόσον, έχουμε εισάγει τα στοιχεία που χρειαζόμαστε πατάμε Next και μας εμφανίζεται το δεύτερο παράθυρο για τις ρυθμίσεις. Τώρα, όπως φαίνεται στην παρακάτω εικόνα θα πρέπει να επιλέξουμε την παράμετρο “This plug-in will make contributions to the UI”, για να βάλουμε στο toolbar και σε μια υποκατηγορία του κυρίως menu, το command που θα εκτελέσει η εφαρμογή μας.



Μετά επιλέγουμε το template “Hello, World Command” για να παραχθούν τα extensions που θα χρειάζεται το plug-in και πατάμε “Finish” για να τελειώσουμε με την δημιουργία του project.



Όταν τελειώσουμε με την δημιουργία του project θα δημιουργηθεί ο κώδικας που θα εκτελείται όταν εκτελούμε το command, καθώς και τα extensions που χρειάζονται για να εμφανιστεί το command στο toolbar και σε μια υποκατηγορία του κυρίως menu. Ο κώδικας που θέλουμε να εκτελεστεί θα πρέπει να βρίσκεται σε μια κλάση η οποία περιέχει μια μέθοδο execute(). Ακολουθούν τα extensions που δημιουργήθηκαν για αυτές τις λειτουργίες και θα επεξηγηθούν περαιτέρω. Στην αριστερή στήλη της παρακάτω εικόνας βρίσκονται όλα τα extensions του project και στην δεξιά στήλη τα στοιχεία για το καθένα από αυτά, εφόσον επιλεγεί κάποιο extension από την αριστερή στήλη.



Στην προηγούμενη εικόνα δημιουργούμε την κατηγορία στην οποία θα υπάρχει το command, της δίνουμε ένα όνομα και κρατάμε το id της, για να το βάλουμε στα στοιχεία του command και δίνουμε και σε αυτό ένα όνομα, όπως φαίνεται στην παρακάτω αριστερή εικόνα. Στην δεξιά εικόνα ρυθμίζουμε την κλάση που θα έχει τον κώδικα που θα εκτελείται, όταν πυροδοτείται το command.

Extension Element Details
Set the properties of 'command'. Required fields are denoted by '*'. Deprecated fields are denoted by '~'.

id*: Plugin_Development.commands.sampleCommand
name*: Sample Command
category():
description:
categoryId: Plugin_Development.commands.category Browse...
defaultHandler: Browse...
returnType: Browse...
helpContext:

Extension Element Details
Set the properties of 'handler'. Required fields are denoted by '*'.

commandId: Plugin_Development.commands.sampleCommand Browse...
class: plugin_development.handlers.SampleHandler Browse...
helpContext:

Ύστερα δημιουργούμε ένα binding στο οποίο δίνουμε την αλληλουχία M1+6, η οποία στο πληκτρολόγιο είναι συντόμευση για τον συνδυασμό των κουμπιών ctrl και 6. Στο commandId θα βάλουμε το id του command που θέλουμε να εκτελεστεί, όταν πατιέται αυτός ο συνδυασμός πλήκτρων. Όμως αν υπάρχει και άλλο command που έχει αυτήν

Extension Element Details
Set the properties of 'key'. Required fields are denoted by '*'.

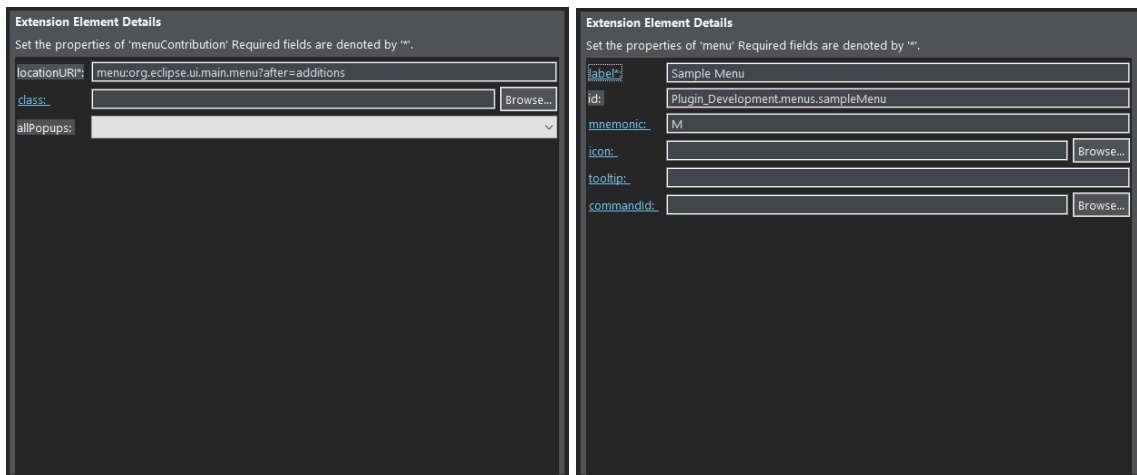
sequence*: M1+6
schemeId: org.eclipse.ui.defaultAcceleratorConfiguration Browse...
contextId: org.eclipse.ui.contexts.window Browse...
commandId: Plugin_Development.commands.sampleCommand Browse...
platform:
locale:

την συντόμευση, το eclipse θα βγάλει ένα menu που θα μας δώσει την επιλογή να διαλέξουμε ποιο command θέλουμε να εκτελέσουμε, όπως φαίνεται στην παρακάτω δεξιά εικόνα.

Analysis	Ctrl+6
Sample Command	Ctrl+6

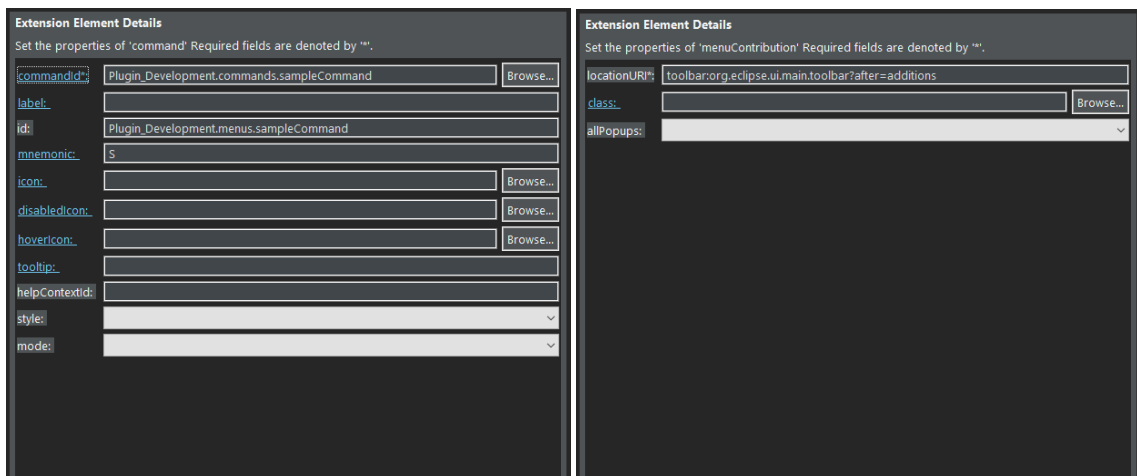
Με την δημιουργία της συντόμευσης έχουμε τελειώσει από τα βασικά που χρειαζόνταν για την εκτέλεση του κώδικα του plug-in. Το μόνο που μένει είναι η δημιουργία των γραφικών στοιχείων, και πιο συγκεκριμένα η εμφάνιση του command σε ειδικά προσαρμοσμένο menu, αλλά και στο toolbar του eclipse.

Στην αριστερή εικόνα που ακολουθεί, στην παράμετρο `locationURI` βάζουμε την τοποθεσία στην οποία θα βρίσκεται το menu για το `command` που φτιάξαμε προηγουμένως. Στην δεξιά εικόνα, στην παράμετρο `label` δίνουμε το όνομα που θα έχει το menu.



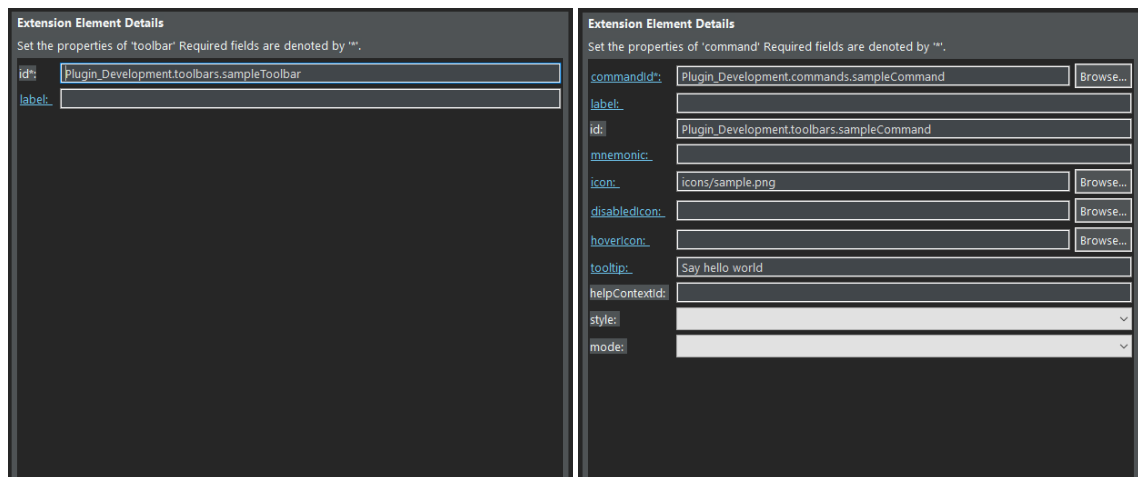
The image shows two side-by-side screenshots of the 'Extension Element Details' dialog box in Eclipse. The left screenshot shows the 'menuContribution' tab with the following fields: `locationURI` (set to 'menu:org.eclipse.ui.main.menu?after=addons'), `class` (empty), and `allPopups` (set to 'true'). The right screenshot shows the 'menu' tab with the following fields: `label` (set to 'Sample Menu'), `id` (set to 'Plugin_Development.menus.sampleMenu'), `mnemonic` (set to 'M'), `icon` (empty), `tooltip` (empty), and `commandId` (empty).

Στην αριστερή επόμενη εικόνα στην παράμετρο `commandId`, βάζουμε το `command` το οποίο περιέχει την μέθοδο `execute()`. Έτσι, ολοκληρώθηκε η δημιουργία του menu για το `command` στο κυρίως menu του eclipse. Στην δεξιά εικόνα, όπως κάναμε και με το menu, βάζουμε στην παράμετρο `locationURI` την τοποθεσία που θα βρίσκεται η επιλογή του `command` στο toolbar.

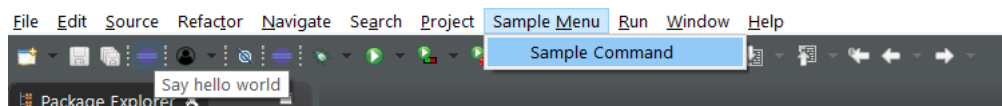


The image shows two side-by-side screenshots of the 'Extension Element Details' dialog box in Eclipse. The left screenshot shows the 'command' tab with the following fields: `commandId` (set to 'Plugin_Development.commands.sampleCommand'), `label` (empty), `id` (set to 'Plugin_Development.menus.sampleCommand'), `mnemonic` (set to 'S'), `icon` (empty), `disabledIcon` (empty), `hoverIcon` (empty), `tooltip` (empty), `helpContextId` (empty), `style` (set to 'default'), and `mode` (set to 'default'). The right screenshot shows the 'menuContribution' tab with the following fields: `locationURI` (set to 'toolbar:org.eclipse.ui.main.toolbar?after=addons'), `class` (empty), and `allPopups` (set to 'true').

Απομένουν άλλα δύο βήματα για να ολοκληρωθεί η δημιουργία των τρόπων, με τους οποίους θα καλείται ο κώδικας του `command handler`. Στην αριστερή εικόνα που ακολουθεί δημιουργούμε μια θέση στο toolbar για το plug-in που δημιουργήσαμε. Στην δεξιά εικόνα, στην παράμετρο `commandId` βάζουμε το `id` που έχει το `command`, στην παράμετρο `tooltip` θα γράψουμε μια συμβολοσειρά η οποία περιγράφει τι κάνει η εκτέλεση του `command` και στην παράμετρο `icon` μπορούμε να βάλουμε κάποια εικόνα που θέλουμε να εμφανίζουμε στην επιλογή που δημιουργήθηκε στο toolbar. Η προεπιλεγμένη τιμή για αυτήν την παράμετρο είναι το "icons/sample.png".



Το τελικό αποτέλεσμα με το οποίο μπορεί να κληθεί για εκτέλεση ο κώδικας του plug-in είναι το εξής. Μια επιλογή στο toolbar και μια επιλογή στο κυρίως menu.



Εξαιτίας του project template που χρησιμοποιήσαμε, όταν πυροδοτηθεί το συγκεκριμένο command θα εκτελέσει τον εξής κώδικα.

```
package plugin_development.handlers;

import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.handlers.HandlerUtil;
import org.eclipse.jface.dialogs.MessageDialog;

public class SampleHandler extends AbstractHandler {

    @Override
    public Object execute(ExecutionEvent event) throws ExecutionException {
        IWorkbenchWindow window = HandlerUtil.getActiveWorkbenchWindowChecked(event);
        MessageDialog.openInformation(
            window.getShell(),
            "Plugin_Development",
            "Hello, Eclipse world");
        return null;
    }
}
```

Έτσι, θα ανοίξει ένα information παράθυρο που θα μας εμφανίζει ένα μήνυμα, όπως φαίνεται στην επόμενη εικόνα.



2.2 Συσταδοποίηση

Η συσταδοποίηση είναι ένας τύπος της μη επιβλεπόμενης μάθησης και είναι η διαδικασία ομαδοποίησης αντικειμένων με τέτοιον τρόπο που τα αντικείμενα που βρίσκονται στην ίδια ομάδα (συστάδα) είναι πιο όμοια μεταξύ τους από ότι με αντικείμενα που βρίσκονται σε άλλες συστάδες. Η συσταδοποίηση χρησιμοποιείται σε πολλά επιστημονικά πεδία της πληροφορικής και αποτελεί μια τεχνική για την στατιστική ανάλυση των δεδομένων. Η ομοιότητα των αντικειμένων εξαρτάται από το πρόβλημα που θέλουμε να επιλύσουμε, καθώς και την μορφή στην οποία βρίσκονται τα αντικείμενα.

2.2.1 Μέθοδοι συσταδοποίησης

Δεν υπάρχουν κριτήρια τα οποία αποδίδουν πάντα μια καλή συσταδοποίηση. Η επιλογή της μεθόδου που θα χρησιμοποιηθεί εξαρτάται από τον χρήστη, το πρόβλημα προς επίλυση, καθώς και τα κριτήρια που ο χρήστης θέλει να ικανοποιήσει. Υπάρχουν τέσσερις κατηγορίες μεθόδων, οι οποίες μπορούν να χρησιμοποιηθούν για την ομαδοποίηση των αντικειμένων σε συστάδες.

- **Μέθοδοι με βάση την πυκνότητα**

Αυτές οι μέθοδοι θεωρούν τις συστάδες, ως πυκνές περιοχές στον χώρο των δεδομένων, οι οποίες διαχωρίζονται από τις υπόλοιπες με αραιές περιοχές. Επιπλέον, έχουν αρκετά καλή ακρίβεια και μπορούν να συγχωνεύσουν δύο συστάδες μεταξύ τους. Οι πιο γνωστές μέθοδοι σε αυτήν την κατηγορία είναι η DBSCAN, καθώς και η OPTICS.

- **Μέθοδοι με βάση την ιεραρχία**

Οι συστάδες που δημιουργούνται σε αυτές τις μεθόδους σχηματίζουν μια δομή που μοιάζει με δέντρο με βάση την ιεραρχία των συστάδων. Οι νέες συστάδες δημιουργούνται με βάση την προηγούμενη συστάδα που είχε σχηματιστεί. Υπάρχουν δύο προσεγγίσεις με τις οποίες σχηματίζονται οι νέες συστάδες. Η προσέγγιση από πάνω προς τα κάτω (divisive) και η προσέγγιση από κάτω προς τα πάνω (agglomerative). Οι πιο γνωστές μέθοδοι σε αυτήν την κατηγορία είναι η CURE, καθώς και η BIRCH. Περισσότερα θα εξηγηθούν σε επόμενο κεφάλαιο.

- **Μέθοδοι με βάση τον διαχωρισμό των αντικειμένων**

Αυτές οι μέθοδοι διαχωρίζουν τα αντικείμενα σε k αριθμό από συστάδες και κάθε διαχωρισμός αποτελεί και από μία συστάδα. Αυτή η κατηγορία μεθόδων

χρησιμοποιείται για την βελτίωση ενός αντικειμενικού κριτηρίου, για παράδειγμα η απόσταση είναι ένα τέτοιο κριτήριο ομοιότητας. Οι πιο γνωστές μέθοδοι σε αυτήν την κατηγορία είναι η k-Means, καθώς και η CLARANS.

- **Μέθοδοι με βάση κάποιο πλέγμα**

Σε αυτήν την κατηγορία μεθόδων ο χώρος των δεδομένων σχηματίζεται σε έναν πεπερασμένο αριθμό από κελιά που σχηματίζουν μια μορφή πλέγματος. Η συσταδοποίηση σε αυτά τα πλέγματα είναι γρήγορη και ανεξάρτητη από τον όγκο των δεδομένων. Οι πιο γνωστές μέθοδοι σε αυτήν την κατηγορία είναι η STING, καθώς και η CLIQUE.

2.2.2 Ιεραρχική συσταδοποίηση

Οι συνήθεις αλγόριθμοι της ιεραρχικής συσταδοποίησης χρησιμοποιούν έναν πίνακα ομοιότητας ή απόστασης των ομάδων για να γίνει ο διαχωρισμός ή η συγχώνευση μιας ομάδας σε κάθε εκτέλεση της επανάληψης του αλγορίθμου. Τα αποτελέσματά της παρουσιάζονται συνήθως σε μορφή δένδροδιαγράμματος, όπου καταγράφονται οι ακολουθίες από συγχωνεύσεις και διαχωρισμούς των ομάδων που υπήρξαν στην συσταδοποίηση. Όπως προαναφέρθηκε, η ιεραρχική συσταδοποίηση χωρίζεται σε δύο κατηγορίες, την από κάτω προς τα πάνω (agglomerative) και την από πάνω προς τα κάτω (divisive):

- **Από κάτω προς τα πάνω προσέγγιση**

Κάθε παρατήρηση ξεκινάει με την δική της συστάδα. Έπειτα, ζευγάρια από συστάδες συγχωνεύονται μεταξύ τους, καθώς μια από αυτές ανεβαίνει πιο ψηλά στην ιεραρχία.

- **Από πάνω προς τα κάτω προσέγγιση**

Όλες οι παρατηρήσεις ξεκινάνε ως μια συστάδα. Έπειτα, διαχωρίζονται διαδοχικά η μία μετά την άλλη, καθώς μια από αυτές κατεβαίνει πιο χαμηλά στην ιεραρχία.

Ο συνήθης αλγόριθμος της από κάτω προς τα πάνω συσταδοποίησης έχει μια χρονική πολυπλοκότητα της τάξης του $O(n^3)$ και απαιτεί μνήμη της τάξης του $O(n^2)$ που καθιστά αυτήν την υλοποίηση πολύ αργή ακόμη και για μεσαίο όγκο δεδομένων. Ωστόσο υπάρχουν μέθοδοι για την από κάτω προς τα πάνω προσέγγιση, όπου για ειδικές περιπτώσεις είναι πιο αποδοτικοί και έχουν πολυπλοκότητα της τάξης του $O(n^2)$. Αυτές οι μέθοδοι είναι γνωστοί ως CLINK (πλήρους δεσμού) και SLINK (μονού δεσμού).

Σε γενικές περιπτώσεις, με την χρήση της στοίβας η πολυπλοκότητα μπορεί να μειωθεί στην τάξη του $O(n^2 \log n)$. Ο πίνακας γειτνίασης χρησιμοποιείται για να αναπαραστήσει τις κορυφές ενός γράφου και εάν αυτές οι κορυφές συνδέονται μεταξύ τους. Όταν, συνδέονται μεταξύ τους, η τιμή σε αυτό το κελί του πίνακα είναι 1, ειδάλλως είναι 0.

Βασικός αλγόριθμος της από κάτω προς τα πάνω συσταδοποίησης

- 1: Υπολογισμός του πίνακα γειτνίασης
- 2: Κάθε σημείο των δεδομένων αποτελεί από μια συστάδα
- 3: Επανάλαβε
- 4: Συγχώνευση των δύο πιο όμοιων(ή κοντινότερων) συστάδων
- 5: Ενημέρωση του πίνακα γειτνίασης
- 6: Μέχρις ότου απομείνει μια συστάδα

Απαιτείται μια μετρική ανομοιότητας μεταξύ των δεδομένων για να αποφασιστεί ποιες συστάδες θα πρέπει να συγχωνευτούν στην από κάτω προς τα πάνω προσέγγιση, καθώς και ποιες συστάδες θα πρέπει να διαχωριστούν στην από πάνω προς τα κάτω. Σε πολλές μεθόδους της ιεραρχικής συσταδοποίησης, αυτό επιτυγχάνεται με την χρησιμοποίηση μιας κατάλληλης μετρικής, και ενός κριτηρίου σύνδεσης. Το κριτήριο σύνδεσης είναι μια συνάρτηση που έχει ως είσοδο δύο παραμέτρους, τις αποστάσεις μεταξύ των ζευγαριών. Η επιλογή της κατάλληλης μετρικής θα επηρεάσει την μορφή των συστάδων, επειδή μερικά στοιχεία μπορεί να είναι πιο κοντά με βάση κάποια α μετρική από ότι με βάση κάποια β μετρική. Για παράδειγμα, σε έναν δυσδιάστατο χώρο η απόσταση μεταξύ δύο σημείων έχει διαφορετικές τιμές ανάλογα με την μετρική που θα χρησιμοποιηθεί. Ας υποθέσουμε πως έχουμε τα σημεία A(0, 2) και B(2, 3), η τιμή της απόστασής τους με βάση την Ευκλείδεια απόσταση είναι $\sqrt{5}$, ενώ με βάση την απόσταση Manhattan είναι 3. Αναφέρονται στον επόμενο πίνακα μερικές μετρικές απόστασης:

Ευκλείδεια Απόσταση	$d(a, b) = \sqrt{\sum_i (a_i - b_i)^2}$
Απόσταση Manhattan	$d(a, b) = \sum_i a_i - b_i $
Μέγιστη Απόσταση	$d(a, b) = \max_i a_i - b_i $

Για αλφαριθμητικά και μη αριθμητικά δεδομένα χρησιμοποιούνται συνήθως μετρικές όπως η απόσταση Hamming, καθώς και ο δείκτης Jaccard για να βρεθεί η ομοιότητα που έχουν τα στοιχεία μεταξύ τους, όταν οι τιμές των κελιών είναι δυαδικές.

Το κριτήριο της σύνδεσης καθορίζει την απόσταση μεταξύ των δύο ζευγαριών με την χρήση κάποιας συνάρτησης. Μερικά από τα πιο συνήθη κριτήρια σύνδεσης μεταξύ δύο σετ παρακολούθησης, του A και του B είναι τα εξής:

Μέγιστο ή Αλγόριθμος πλήρους δεσμού (CLINK)	$\max \{ d(a, b) : a \in A, b \in B \}$
Ελάχιστο ή Αλγόριθμος μονού δεσμού (SLINK)	$\min \{ d(a, b) : a \in A, b \in B \}$

3 JavaModel και AST

Το JavaModel και το AST αποτελούν βιβλιοθήκες του Eclipse και χρησιμοποιήθηκαν για την εύρεση των στοιχείων των συστατικών που έχει το project που θέλουμε να αναλύσουμε. Οι δύο αυτές βιβλιοθήκες είναι κόμματα των Java Development Tools (JDT) του Eclipse και είναι ορισμένες στο πακέτο «org.eclipse.jdt.core». Επίσης, θα χρησιμοποιηθούν και άλλες κλάσεις (όπως η IWorkspace, η IWorkspaceRoot, καθώς και η IProject), οι οποίες βρίσκονται στο πακέτο «org.eclipse.core.resources», καθώς και επιπλέον κλάσεις (όπως η IMethodBinding και η ITypeBinding) που βρίσκονται στο πακέτο «org.eclipse.jdt.core.dom». Καταρχάς προτού περάσουμε στην εξήγηση του JavaModel και του AST θα πρέπει να αναφερθεί ο τρόπος με τον οποίο εξάγουμε τα projects από το Eclipse.

```
IWorkspaceRoot root = ResourcesPlugin.getWorkspace().getRoot();
IProject[] projects = root.getProjects();
```

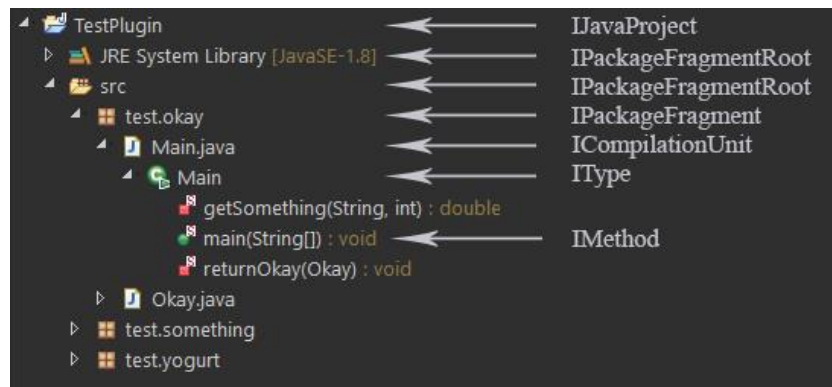
Η πρώτη γραμμή του παραπάνω κώδικα επιστρέφει στο αντικείμενο root το workspace το οποίο έχει το Eclipse. Έπειτα η δεύτερη γραμμή του κώδικα επιστρέφει τα projects που βρίσκονται στο IDE του Eclipse αυτήν την στιγμή που καλείται. Όμως θα χρησιμοποιήσουμε μόνο όσα projects είναι ανοιχτά εκείνη την στιγμή και μόνο εκείνα που είναι Java Projects. Έτσι θα ελέγξουμε κάθε project για τα παραπάνω με τον εξής κώδικα:

```
for (IProject project : projects) {
    if(project.isOpen()) {
        try {
            if (project.isNatureEnabled(JDT_NATURE)) {
                // Γίνεται ανάλυση των δομικών στοιχείων κάθε project
                analyseMethods(project);
            }
        } catch (CoreException e) {
            e.printStackTrace();
        }
    }
}
```

Ο τρόπος με τον οποίο λειτουργεί η μέθοδος analyseMethods(project) και πραγματοποιεί την ανάλυση των δομικών στοιχείων θα αναφερθεί σε παρακάτω υποκεφάλαιο.

3.1 JavaModel

Κάθε project της Java παρουσιάζεται εσωτερικά από ένα μοντέλο, το οποίο είναι ελαφρύ από άποψη χρήσης πόρων και ανθεκτικό σε λάθη. Δεν περιέχει τόσες πληροφορίες όσες το AST, αλλά δημιουργείται πιο γρήγορα από εκείνο. Παρακάτω θα αναφερθούν οι εντολές που χρησιμοποιήθηκαν από το συγκεκριμένο πακέτο, ο λόγος που χρησιμοποιήθηκαν και ο τρόπος σύνταξής τους.



Εικόνα 1: Οι πληροφορίες ενός project για το JavaModel του.

- Το IJavaProject είναι ένας κόμβος του JavaModel, αντιπροσωπεύει ένα Java Project και περιέχει ως παιδιά κόμβους, τα IPackageFragmentRoots.
- Το IPackageFragmentRoot μπορεί να είναι ένας φάκελος με κλάσεις, ένα .zip ή ένα .jar αρχείο.
- Το IPackageFragment είναι ένα πακέτο και περιέχει ένα ή περισσότερα ICompilationUnit, εάν το IPackageFragment είναι πηγαίος κώδικας, και IClassFile, εάν το IPackageFragment είναι σε δυαδική μορφή. Τα IPackageFragments δεν ακολουθούν την δομή γονέα-παιδιού.
- Το ICompilationUnit είναι ένα αρχείο πηγαίου κώδικα γραμμένο σε Java.
- Τα IMethod και IType αποτελούν παιδιά του ICompilationUnit.

3.1.1 IPackageFragment

Ένα IPackageFragment μπορεί να περιέχει ένα πακέτο το οποίο βρίσκεται σε ένα project με την εξής εντολή:

```
IPackageFragment[] packages = JavaCore.create(project).getPackageFragments();
```

Τα πακέτα αυτά εξάγονται με την χρήση της μεθόδου `getPackageFragments()` από κάποιο `IProject` του πίνακα `projects` που προαναφέρθηκε στην [παράγραφο 3](#). Επιπλέον χρησιμοποιείται η μέθοδος `getElementName()`, όπου επιστρέφει το όνομα που έχει ένα συγκεκριμένο πακέτο, με τον εξής απλό τρόπο:

Το `mypackage` αποτελεί κομμάτι του παραπάνω πίνακα `packages`.

```
String packageName = mypackage.getElementName();
```

3.1.2 ICompilationUnit

Το `ICompilationUnit` μπορεί να περιέχει τον κώδικα ενός `.java` αρχείου και συγκεκριμένα στην περίπτωση μας των `.java` αρχείων που βρίσκονται σε ένα συγκεκριμένο πακέτο χρησιμοποιώντας την μέθοδο `getCompilationUnits()` της κλάσης `IPackageFragment`. Και συντάσσεται με τον παρακάτω τρόπο:

```
ICompilationUnit[] units = mypackage.getCompilationUnits();
```

Το `mypackage` είναι ένα πακέτο και αποτελεί υπομέρος του πίνακα `packages` που αναφέρθηκε στο [παράρτημα 3.1.1](#). Αυτή η διαδικασία γίνεται για κάθε πακέτο του `project` που αναλύουμε οπότε έπειτα με χρήση επαναληπτικής δομής παίρνουμε τα δομικά στοιχεία του πακέτου.

3.1.3 IMethod

Η κλάση `IMethod` χρησιμοποιείται, γιατί η βιβλιοθήκη «CallHierarchy» του Eclipse που μας δίνει τις μεθόδους που καλούνται από κάποια άλλη μέθοδο χρησιμοποιεί αντικείμενα τύπου `IMethod`. Θα αναφερθεί εκτενέστερα ο τρόπος που χρησιμοποιείται η συγκεκριμένη βιβλιοθήκη, στην [ενότητα 4](#).

3.1.4 IMethodBinding

Ένα `method binding` συνήθως αντιστοιχείται άμεσα σε μια μέθοδο ή μια δήλωση ενός κατασκευαστή στον πηγαίο κώδικα. Σε αυτήν την εργασία χρησιμοποιούμε

```
IMethodBinding mBinding = method.resolveBinding();
```

την μέθοδο `resolveBinding()` για την μετατροπή ενός αντικειμένου `MethodDeclaration` σε αντικείμενο τύπου `IMethod`.

Το αντικείμενο `method` στο παραπάνω παράδειγμα είναι τύπου `MethodDeclaration`.

Επίσης χρησιμοποιούμε την `getDeclaringClass()` για να πάρουμε την κλάση στην οποία βρίσκεται αυτή η μέθοδος και πιο συγκεκριμένα το όνομά της:

```
mBinding.getDeclaringClass().getName().toString();
```

Την μέθοδο `getParameterTypes()` που επιστρέφει έναν πίνακα τύπου `ITypeBinding` με τις παραμέτρους της μεθόδου του `mBinding`, καθώς και την μέθοδο `getJavaElement()` που επιστρέφει ένα στοιχείο της Java που αντιστοιχεί στο binding από το οποίο καλέστηκε. Επιστρέφει `null` όταν αυτό το binding δεν αντιστοιχεί σε κάποιο στοιχείο της Java.

3.1.5 ITypeBinding

Ένα type binding αντιπροσωπεύει έναν πλήρως επιλυμένο τύπο. Μπορούν να υπάρχουν πολλά είδη από type bindings:

- Class και Interface: αντιπροσωπεύουν την δήλωση κλάσης που μπορεί να έχει και παραμέτρους τύπου.
- Enum: αντιπροσωπεύει την enum δήλωση.
- Annotation: αντιπροσωπεύει την δήλωση ενός σχόλιου.
- Array type: ο πίνακας τύπων αναφέρεται, αλλά δεν είναι ρητά δηλωμένος.
- Null type: ειδικό είδος τύπου του `null`.
- Τύπος μεταβλητής: αντιπροσωπεύει την δήλωση ενός τύπου μεταβλητής, πιθανότατα με τα όρια του τύπου.
- Wildcard type: αντιπροσωπεύει έναν wildcard που χρησιμοποιείται ως δήλωση τύπου σε μια παραμετροποιήσιμη αναφορά τύπου.
- Raw type: αντιπροσωπεύει μια παραμετροποιήσιμη αναφορά σε έναν γενικό τύπο.
- Παραμετροποιήσιμος τύπος: αντιπροσωπεύει μια αντιγραφή μιας δήλωσης τύπου με αντικαταστάτες για κάθε τύπο παραμέτρου που είχε η δήλωση.
- Capture: αντιπροσωπεύει ένα capture binding.

Ανάλογα με την μέθοδο που καλείται στο method binding, το type binding μπορεί να έχει έναν τύπο από τους προαναφερθέντες. Στην συγκεκριμένη εργασία όπως αναφέρθηκε

```
ITypeBinding[] bindCallee = mBinding.getParameterTypes();
```

συνολικά στην [ενότητα 3.1.4](#), χρησιμοποιείται η κλάση `ITypeBinding` για να μας δώσει έναν πίνακα με τους τύπους των παραμέτρων από το `IMethodBinding` της μεθόδου στην οποία καλέστηκε.

3.2 AST

Το AST (Abstract Syntax Tree) είναι μια λεπτομερής δεντρική απεικόνιση ενός πηγαίου κώδικα γραμμένου σε Java. Το AST του Eclipse αποτελεί μια βιβλιοθήκη, η οποία σου δίνει την δυνατότητα να διαβάσεις, να διαγράψεις και να τροποποιήσεις τον πηγαίο κώδικα ενός αρχείου Java. Με την χρήση του AST μπορούμε να περιηγηθούμε από έναν γονικό κόμβο σε έναν κόμβο παιδί και το αντίστροφο. Ένα AST λόγω της δεντρικής δομής που έχει, δεν μπορεί να περιέχει βρόγχους και κυκλικούς γράφους. Επιπλέον δεν περιέχει υποδέντρα, τα οποία να μπορούν να λείπουν από αυτό. Ωστόσο εάν ένας κόμβος απαιτείται να έχει μια συγκεκριμένη ιδιότητα, τότε πάντα παρέχεται μια συντακτικά ορθή αρχική τιμή. Κάθε στοιχείο του κώδικα της Java αναπαρίσταται από κάποιο ASTNode. Τα AST υποστηρίζουν το visitor pattern με την χρήση της κλάσης ASTVisitor, επίσης μπορεί να χρησιμοποιηθεί η κλάση NodeFinder για να βρεθεί ένας συγκεκριμένος κόμβος μέσα στο δέντρο. Το visitor pattern είναι ένας τρόπος με τον οποίο διαχωρίζεται ο αλγόριθμος από την δομή του αντικειμένου που αυτός ενεργεί. Ένα πρακτικό αποτέλεσμα αυτής της διαχώρισης είναι η ικανότητα να προστίθενται νέες ενέργειες σε ήδη υπάρχουσες δομές αντικειμένου χωρίς να τροποποιούνται οι δομές τους.

3.2.1 ASTVisitor

Κάθε ASTNode επιτρέπει την δημιουργία ερωτημάτων χρησιμοποιώντας έναν Visitor. Κάθε υποκλάση του ASTNode μπορεί να περιέχει τέσσερις μεθόδους, την visit, την endVisit(), την preVisit(), καθώς και την postVisit(). Οι μέθοδοι που προαναφέρθηκαν δέχονται ως παράμετρο έναν κόμβο και επισκέπτονται αυτόν τον κόμβο για να εκτελέσουν μια αυθαίρετη διεργασία. Επιπλέον, οι προεπιλεγμένες εφαρμογές τους που παρέχονται από την κλάση ASTVisitor επιστρέφουν true, με μόνη εξαίρεση όταν καλείται με τον τύπο Javadoc του ASTNode που επιστρέφει false. Η μέθοδος visit() θα επιστρέψει μια τιμή boolean, η οποία θα είναι false εάν δε θα επισκεφτεί τα παιδιά κόμβους του συγκεκριμένου κόμβου και true εάν επισκεφτεί και τα παιδιά κόμβους, αφότου επισκεφτεί τον συγκεκριμένο κόμβο που είχε ως παράμετρο. Η endVisit() όταν χρησιμοποιείται με τον παραδοσιακό τρόπο, καλείται εφόσον έχει τελειώσει η επίσκεψη όλων των κόμβων παιδιών ή αμέσως, όταν η μέθοδος visit() επιστρέψει τιμή false. Η preVisit() καλείται πριν την μέθοδο visit() και η postVisit() καλείται με την μέθοδο endVisit(). Ωστόσο και

οι δύο διαφέρουν ανάλογα με τον τύπο `ASTNode` που δέχτηκαν ως παράμετρο η `visit()` και η `endVisit()` αντίστοιχα.

3.2.2 ASTNode

Το `ASTNode` είναι μια abstract υπερκλάση όλων των τύπων που μπορεί να έχει ένας κόμβος του AST και παρέχει συγκεκριμένες πληροφορίες σχετικά με το αντικείμενο που αναπαριστά. Όταν ένας κόμβος του AST αποτελεί μέρος αυτού μπορεί να έχει μόνο έναν γονικό κόμβο. Κάθε `ASTNode` μπορεί να έχει διάφορους τύπους όπως εκφράσεις (`Expression`), ονόματα (`Name`, ένα υποσύνολο του `Expression`), δηλώσεις (`Statement`), τύπους (`Type`), καθώς και μια δήλωση ενός κομματιού κώδικα (`BodyDeclaration`). Από την κλάση `ASTNode` χρησιμοποιείται η μέθοδος `accept()` όπου παίρνει ως παράμετρο ένα αντικείμενο τύπου `ASTVisitor`, έτσι αποδέχεται την επίσκεψη αυτού του αντικειμένου στον κόμβο που βρίσκεται αυτήν την στιγμή, για να αρχίσει η ανάλυση του.

3.2.3 ASTParser

Για κάθε `ICompilationUnit` χρησιμοποιείται η κλάση `ASTParser`, όπου δημιουργείται ένα AST και επιστρέφεται ένα `ASTNode` και πιο συγκεκριμένα στην δική μας περίπτωση ένα `CompilationUnit`.

```
CompilationUnit parse = parse(unit);

private static CompilationUnit parse(ICompilationUnit unit) {
    ASTParser parser = ASTParser.newParser(AST.JLS10);
    parser.setKind(ASTParser.K_COMPILATION_UNIT);
    parser.setSource(unit);
    parser.setResolveBindings(true);
    return (CompilationUnit) parser.createAST(null); // parse
}
```

Χρησιμοποιούνται οι μέθοδοι `setKind()` όπου παίρνει ως παράμετρο μεθόδου το είδος του αντικειμένου που θέλουμε να κάνει `parse`, η `setSource()` όπου παίρνει ως παράμετρο τον πηγαίο κώδικα που πρέπει να κάνει `parse`, συνήθως είναι όπως και στην περίπτωσή μας ένα `ICompilationUnit`, καθώς και η μέθοδος `setResolveBindings()` που μπορεί να πάρει τιμές `true` ή `false`, ανάλογα με το αν θέλουμε ή όχι να παρέχει πληροφορίες για τα bindings των AST Nodes που θα δημιουργήσει. Έπειτα κάνουμε cast το AST Node που δημιουργήθηκε σε `CompilationUnit` και το επιστρέφουμε.

3.2.4 NodeFinder

Με την χρήση της κλάσης `NodeFinder` πραγματοποιούμε εύρεση ενός συγκεκριμένου κόμβου χρησιμοποιώντας την μέθοδο `perform()`, η οποία δέχεται δυο παραμέτρους. Η πρώτη παράμετρος είναι τύπου `ASTNode` και η δεύτερη είναι τύπου `ISourceRange`, όπου η τελευταία περιέχει το εύρος του πηγαίου κώδικα από τον οποίο θέλουμε να επιστρέψουμε το `ASTNode`. Πιο συγκεκριμένα, στην περίπτωση μας χρειάστηκε να χρησιμοποιηθεί η μέθοδος `perform()` για να μετατρέψουμε ένα αντικείμενο τύπου `IMethod` σε `MethodDeclaration`. Στο παράδειγμα που ακολουθεί η μεταβλητή `method` είναι τύπου `IMethod`, έτσι με την πρώτη γραμμή του παραδείγματος παίρνουμε τον πηγαίο κώδικα του `method` και το μετατρέπουμε σε `CompilationUnit`, κάνοντάς το `parse` με την βοήθεια του `ASTParser`. Έπειτα χρησιμοποιούμε το `unit` ως πρώτη παράμετρο στην μέθοδο, και ως δεύτερη το εύρος του κώδικα που έχει η μέθοδος. Έτσι επιστρέφεται ένα `ASTNode` για την συγκεκριμένη μέθοδο, το οποίο κάνουμε `cast` σε `MethodDeclaration`. Η μετατροπή του `IMethod` σε `MethodDeclaration` χρειάστηκε, επειδή το AST μας παρέχει

```
ICompilationUnit methodCompilationUnit = method.getCompilationUnit();  
CompilationUnit unit = parse(methodCompilationUnit);  
  
ASTNode currentNode = NodeFinder.perform(unit, method.getSourceRange());  
  
return (MethodDeclaration) currentNode;
```

περισσότερες πληροφορίες για τα δομικά στοιχεία των μεθόδων από ότι η κλάση `IMethod` του `JavaModel`, έτσι αποθηκεύουμε τις μεθόδους που επισκέπτονται από το `ASTVisitor` σε ένα `ArrayList` από `MethodDeclarations`.

4 CallHierarchy

Η «CallHierarchy» είναι μια εσωτερική βιβλιοθήκη του Eclipse, βρίσκεται στο πακέτο «org.eclipse.jdt.internal.corext.callhierarchy» και μας βοήθησε στην εύρεση των μεθόδων που καλούν άλλες μεθόδους, δηλαδή στην εύρεση των συσχετίσεων μεταξύ των κλάσεων. Από αυτό το πακέτο χρησιμοποιήθηκαν η κλάση «CallHierarchy» για την εύρεση μιας λίστας από μεθόδους, οι οποίες καλούνται από μια συγκεκριμένη μέθοδο, καθώς και η κλάση «MethodWrapper» για να επιστραφούν τα αντικείμενα τύπου IMethod για την προαναφερθείσα λίστα.

4.1 Κλάση CallHierarchy

Η κλάση «CallHierarchy» χρησιμοποιήθηκε για να βρεθούν στατικά οι κλήσεις των μεθόδων από άλλες μεθόδους, χωρίς την εκτέλεση του project που θέλουμε να αναλύσουμε. Για να πάρουμε τις πληροφορίες που χρειαζόμαστε πρέπει πρώτα να δημιουργηθεί ένα αντικείμενο αυτής της κλάσης, με σκοπό να χρησιμοποιηθούν οι εξής μέθοδοι:

- **getDefault()**

Με την κλήση αυτής της στατικής μεθόδου δημιουργείται ένα νέο instance, εφόσον δεν υπάρχει ήδη κάποιο ενεργό και χρησιμοποιείται συνήθως για την δημιουργία αντικειμένων αυτής της κλάσης.

- **setFilterEnabled(boolean value)**

Ανάλογα την τιμή που έχει η value, ενεργοποιεί (true) ή απενεργοποιεί (false) την χρήση των φίλτρων.

- **setFilters(String value)**

Με την χρήση της μεθόδου setFilters() μπορούμε να θέσουμε στην αλφαριθμητική μεταβλητή value ποια πακέτα του project δε θα λάβει υπόψιν στην διαδικασία της εύρεσης των συσχετίσεων.

- **getCallerRoots(IMember[] values)**

Επιστρέφει έναν πίνακα από αντικείμενα τύπου MethodWrapper, ώστε έπειτα με την χρήση της μεθόδου getCalls() να βρεθούν οι κλήσεις των μελών του πίνακα και πιο συγκεκριμένα οι μέθοδοι που καλούν την συγκεκριμένη μέθοδο.

4.2 Κλάση MethodWrapper

Η κλάση «MethodWrapper» χρησιμοποιήθηκε για να βρεθούν οι κλήσεις των μεθόδων. Ένα αντικείμενο αυτής της κλάσης απεικονίζει τα γενικά στοιχεία μιας κλήσης μεθόδου (είτε από, είτε προς την μέθοδο) και μας δίνει την δυνατότητα να αποκτήσουμε πληροφορίες για αυτές τις μεθόδους που την καλούν. Το IProgressMonitor είναι ένα interface που με την δημιουργία ενός αντικειμένου ελέγχει την πρόοδο που υπάρχει σε μια δραστηριότητα. Χρησιμοποιήθηκε μόνο μια μέθοδος από αυτήν την κλάση, με όνομα getCalls(), η οποία δέχεται μόνο μια παράμετρο τύπου IProgressMonitor και επιστρέφει έναν πίνακα από αντικείμενα τύπου MethodWrapper που καλούν το MethodWrapper για το οποίο καλέστηκε η μέθοδος getCalls().


























5 R

Η R είναι μια γλώσσα προγραμματισμού και ένα πακέτο στατιστικής επεξεργασίας ανοιχτού λογισμικού που επιτρέπει σε κάποιον χρήστη να κάνει υπολογιστική στατιστική και δημιουργία στατιστικών γραφημάτων. Η R είναι μια γλώσσα ανοιχτού κώδικα, η οποία είναι ως επί το πλείστον γραμμένη σε C, Fortran, καθώς και την ίδια την R. Έχει επηρεαστεί πολύ από δύο άλλες γλώσσες προγραμματισμού, την S που αναπτύχθηκε κυρίως από τον John Chambers και από τους Rick Becker και Allan Wilks των Εργαστηρίων της Bell, καθώς και την γλώσσα προγραμματισμού Scheme η οποία αναπτύχθηκε από τους Guy L. Steele και Gerald Jay Sussman που ήταν προγραμματιστές του MIT. Έτσι καταλήξαν σε μια γλώσσα η οποία είναι αρκετά παρόμοια σε εμφάνιση με την S, και η υποκείμενη εφαρμογή και η σημασιολογία της έχουν προέλθει από την Scheme. Το πλεονέκτημα της R είναι ότι κάνει τα ίδια πράγματα με την S, αλλά χρησιμοποιώντας λιγότερο κώδικα και επιτρέπει την αλληλεπίδραση με άλλες γλώσσες (π.χ. C/C++, Python, Java), με αρχεία δεδομένων (π.χ. Excel), καθώς και με άλλα στατιστικά πακέτα (π.χ. SPSS, Matlab). Το μειονέκτημα της R είναι ότι καταναλώνει πολύ μνήμη και είναι αργή γλώσσα ως προς τον χρόνο εκτέλεσης των εντολών, άρα καλό είναι να μην χρησιμοποιείται για ανάλυση μεγάλου όγκου δεδομένων.

Ο κύριος λόγος που χρησιμοποιήθηκε η γλώσσα R, είναι για την εμφάνιση της γραφικής παράστασης. Μια γραφική παράσταση δημιουργείται από την χρήση της συνάρτησης plot. Η συνάρτηση plot δέχεται τις εξής παραμέτρους:

- Μπορεί να περιέχει μία ή δύο παραμέτρους, οι οποίες μπορεί να είναι είτε το αποτέλεσμα μιας συνάρτησης, είτε πίνακες με αριθμούς. Αυτοί οι παράμετροι είναι αυτό που θέλουμε να εμφανίσουμε στην γραφική παράσταση.
- Η xlab και η ylab παίρνουν ως τιμή μια σειρά από αλφαριθμητικούς χαρακτήρες και είναι οι λεζάντες του άξονα x και y, αντίστοιχα. Η απόδοση τιμής σε αυτές τις δυο παραμέτρους είναι προαιρετική, όμως οι προεπιλεγμένες τιμές είναι x για τον άξονα x και y για τον άξονα y.
- Η main παίρνει και αυτή ως τιμή μια σειρά από αλφαριθμητικούς χαρακτήρες και είναι ο τίτλος της γραφικής παράστασης. Η απόδοση τιμής σε αυτήν την παράμετρο είναι προαιρετική, αλλά η προεπιλεγμένη τιμή είναι μια κενή συμβολοσειρά.

- Η παράμετρος `cx` δέχεται αριθμητικές τιμές, οι οποίες μπορούν να έχουν δεκαδικά ψηφία και είναι το μέγεθος που θα έχει το κάθε σημείο.
- Η παράμετρος `pch` παίρνει κάποια τιμή από τον παρακάτω πίνακα, καθώς και οποιονδήποτε UTF-8 χαρακτήρα.

1: 	6: 	11: 	16: 	21: 
2: 	7: 	12: 	17: 	22: 
3: 	8: 	13: 	18: 	23: 
4: 	9: 	14: 	19: 	24: 
5: 	10: 	15: 	20: 	25: 

Πίνακας με τις τιμές που μπορεί να πάρει το `pch`

Οι τιμές από 21 μέχρι και 25 πρέπει να έχουν και τιμή στην παράμετρο `bg`, αν δεν έχουν τιμή, τότε ως εσωτερικό γέμισμα έχουν το χρώμα άσπρο.

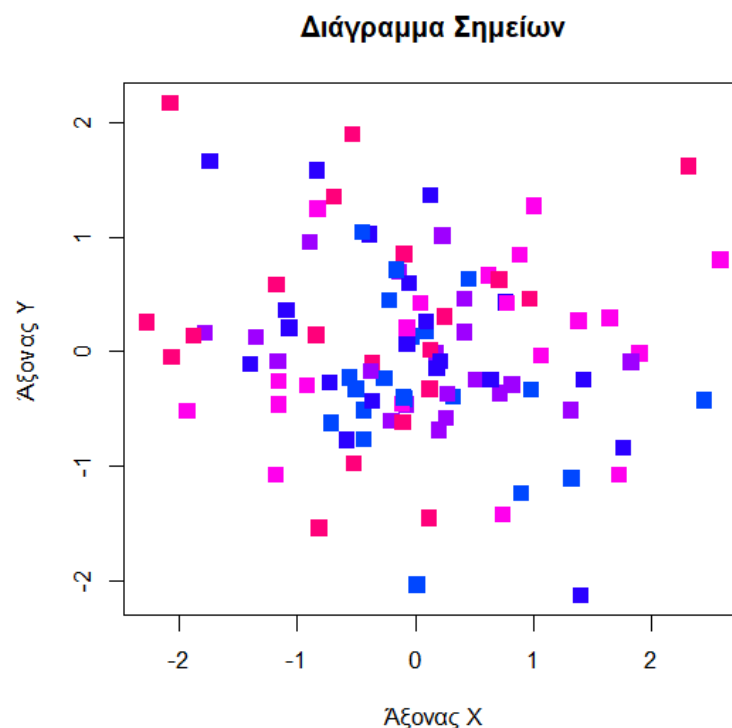
Επίσης, μπορούν να χρησιμοποιηθούν και αλφαριθμητικοί χαρακτήρες, παραδείγματος χάριν ένα γράμμα του ελληνικού αλφαβήτου ('α'). Έτσι, σε κάθε σημείο της γραφικής παράστασης θα εμφανίζεται το γράμμα 'α'.

- Οι παράμετροι `col` και `bg` δέχονται αλφαριθμητικές τιμές από την συνάρτηση `colors()`, η οποία περιέχει εκατοντάδες χρώματα. Ωστόσο, μπορούν να πάρουν και τιμή από την συνάρτηση `rgb()`, η οποία δέχεται τρεις παράμετρους με δεκαδικές τιμές στο διάστημα $[0,1]$. Επίσης, μπορεί να χρησιμοποιηθεί και η συνάρτηση `rainbow()`, αν θέλουμε τα σημεία να έχουν περισσότερα από ένα χρώματα. Η `rainbow()` δέχεται ως παράμετρο μια ακέραια τιμή, η οποία θα αποτελεί τον αριθμό των χρωμάτων που θα έχουν τα σημεία της γραφικής παράστασης. Επιπλέον μια άλλη εκδοχή της συνάρτησης `rainbow()`, μας επιτρέπει να χρησιμοποιούμε ένα συγκεκριμένο εύρος για τα χρώματα. Αυτή η μέθοδος λοιπόν, περιέχει δύο επιπλέον παραμέτρους σε σχέση με την προηγούμενη εκδοχή της. Η δεύτερη παράμετρος και η τρίτη παράμετρος, `start` και `end` αντίστοιχα, δέχονται δεκαδικές τιμές στο διάστημα $[0,1]$. Εάν αυτές οι τιμές είναι προς τις αρχές του εύρους, τότε τα χρώματα των σημείων θα είναι προς το κόκκινο και το κίτρινο, εάν οι τιμές αυτές βρίσκονται στην μέση του εύρους,

τότε θα παραχτεί μια γραφική παράσταση με πράσινο και μπλε χρώμα, ενώ αν βρίσκονται στο τέλος του εύρους θα παραχτεί μια γραφική παράσταση με μπλε και μωβ χρώμα. Η παράμετρος `col` αποτελεί το χρώμα των σημείων στην γραφική παράσταση και η παράμετρος `bg` αποτελεί το «γέμισμα» των σημείων, όταν η `pch` παίρνει τιμές από 21 μέχρι και το 25.

Στο παράδειγμα που ακολουθεί η `set.seed` μας επιτρέπει κάθε φορά που θα τρέχουμε την `rmnorm` να μας δίνει τους ίδιους αριθμούς. Με την χρήση της `rmnorm` παίρνουμε 100 τυχαίους σταθερούς (λόγω του `set.seed`) αριθμούς. Και μετά εμφανίζουμε μια γραφική παράσταση με τα σημεία.

```
set.seed(100)
x <- rnorm(100)
y <- rnorm(100)
plot(x, y, pch=15, col=rainbow(5, start=0.62, end=0.92), cex=1.5,
     xlab='Άξονας Χ', ylab='Άξονας Υ', main='Διάγραμμα Σημείων')
```



5.1 RCaller

Το RCaller είναι μια αυτόνομη βιβλιοθήκη ανοιχτού λογισμικού, γραμμένη στην γλώσσα προγραμματισμού Java και δε βασίζεται σε άλλες εξωτερικές βιβλιοθήκες. Αναπτύχθηκε από τον M. Hakan Satman για να απλοποιηθεί το κάλεσμα εντολών της R στην Java και το μόνο που χρειάζεται για να εκτελεσθεί σε ένα σύστημα είναι η εγκατάσταση αυτών των δύο γλωσσών. Αν και δεν είναι ο πιο αποδοτικός τρόπος για να καλεστεί κώδικας της R από την Java, είναι πολύ απλός και εύκολος για να τον μάθει κάποιος. Απλοποιεί τον τρόπο με τον οποίο επικοινωνούν οι δύο αυτές γλώσσες και επιτρέπει την δημιουργία μεταβλητών που είναι κοινές και για τις δύο γλώσσες. Οι εντολές εκτελούνται σειριακά, αλλά η απόδοση δεν μειώνεται. Η γλώσσα R μπορεί να αξιοποιήσει μόνο ένα thread, όμως με την χρήση του RCaller μπορούν να υπάρξουν πολλαπλές παράλληλες εκτελέσεις στην Java.

Το RCaller βασικά περιτυλίγει όλες τις λεπτομέρειες και εκτελεί μετατροπές τύπων μεταξύ των δύο γλωσσών. Το RCaller μπορεί να χρησιμοποιηθεί με τρεις τρόπους για να εκτελεστεί κώδικας της R μέσω της Java:

- 1 Τα δεδομένα μεταφέρονται, υπολογίζεται ένα διάνυσμα από αποτελέσματα και το αποτέλεσμα αυτού, το διαχειρίζεται η Java. Στο παρασκήνιο δημιουργείται και εκτελείται μια διεργασία του Rscript. Αυτός ο τρόπος είναι χρήσιμος σε περιπτώσεις που πρέπει μια φορά απλά να αποσταλούν τα δεδομένα, να παράγουν κάποιο αποτέλεσμα, όπου μετά αυτό το αποτέλεσμα θα επιστραφεί.
- 2 Μια εξωτερική διεργασία της R δημιουργείται και κρατιέται στην μνήμη χωρίς να τερματιστεί. Εντολές, κλήσεις συναρτήσεων, και δεδομένα αποστέλλονται στην R, τα αποτελέσματα επιστρέφονται πίσω στην Java και η διεργασία που δημιουργήθηκε δεν τερματίζεται, για να χρησιμοποιηθεί ξανά. Αυτός ο τρόπος είναι χρήσιμος σε περιπτώσεις επανάληψης και όταν απαιτούνται διαδραστικοί υπολογισμοί.
- 3 Η διαδικασία κλήσης της R από την Java είναι πιο αφηρημένη και περιτυλίγεται από ένα scripting API που είναι καθορισμένο από το πρότυπο JSR223. Το scripting API εξυπηρετεί έναν απλό τρόπο για την ενσωμάτωση μιας scripting γλώσσας, όπως η javascript με την Java, στην οποία οι κλήσεις συναρτήσεων και οι μεταφορές των δεδομένων είναι καλυμμένες.

Στην συγκεκριμένη εργασία χρησιμοποιείται ο πρώτος τρόπος, γιατί χρειάζεται να στείλουμε στην R μόνο μια φορά τα δεδομένα των συσχετίσεων των πακέτων και σαν

αποτέλεσμα θα μας ανοίξει ένα παράθυρο με την γραφική παράσταση του διαγράμματος που έβγαλε, στην περίπτωση μας επιστρέφει ένα διάγραμμα ιεραρχικής συσταδοποίησης. Οι πιο πολλές κλάσεις του RCaller βρίσκονται στο πακέτο «com.github.rcaller.rstuff».

5.2 Μεταφορά των δεδομένων

Η κλάση RCaller περιέχει μεθόδους για δημιουργία εξωτερικών διεργασιών και για μεταφορά δεδομένων. Τα δεδομένα που μεταφέρονται από την Java, κωδικοποιούνται σε κώδικα της R και οι υπολογισμοί γίνονται από την R. Όταν το αποτέλεσμα καλείται από την Java, μετατρέπεται πρώτα σε έναν ορθό συντακτικά XML κώδικα και έπειτα αναλύεται από την ίδια την Java. Η κλάση RCode περιέχει μεθόδους για μετατροπή των τύπων των δεδομένων, καθώς και μεθόδους για δημιουργία κώδικα της R.

Για να γίνει η μεταφορά των δεδομένων θα πρέπει πρώτα να δημιουργηθούν αντικείμενα των κλάσεων RCaller και RCode.

```
RCaller caller = RCaller.create();  
RCode code = RCode.create();
```

Η RCode παρέχει τις εξής μεθόδους για μεταφορά μονόμετρων δεδομένων από την Java προς την R:

- addLogical(String, boolean) ή addBoolean(String, boolean)

```
boolean temp = false;  
code.addLogical("tempvalue", temp);
```

- addDouble(String, double)

```
double temp = 2.3;  
code.addDouble("tempvalue", temp);
```

- addFloat(String, float)

```
float temp = 2.3f;  
code.addFloat("tempvalue", temp);
```

- addShort(String, short)

```
short temp = 32;  
code.addShort("tempvalue", temp);
```

- addInt(String, int)

```
int temp = 32780;  
code.addInt("tempvalue", temp);
```

- addLong(String, long)

```
long temp = 2147483650;
code.addLong("tempvalue", temp);
```

- addString(String, String)

```
String temp = "nice";
code.addString("tempvalue", temp);
```

Κάθε μία από τις παραπάνω μεθόδους δέχονται δύο παραμέτρους. Η πρώτη παράμετρος είναι το όνομα που θα έχει η μεταβλητή στην R και είναι τύπου String, ενώ η δεύτερη παράμετρος μπορεί να είναι μια μεταβλητή της Java ή και μια συγκεκριμένη τιμή ανάλογα με την μέθοδο που χρησιμοποιήθηκε. Παραδείγματος χάριν ας πούμε ότι χρησιμοποιούμε το παράδειγμα με την μέθοδο addInt(String, int). Η δεύτερη παράμετρος παίρνει την τιμή του temp, δηλαδή την τιμή 32780 και την συσχετίζει με μια μεταβλητή με το όνομα “tempvalue”. Έτσι κάθε φορά που θα καλείται στην R η μεταβλητή “tempvalue”, θα εμφανίζεται η τιμή με την οποία συσχετίστηκε στην Java, δηλαδή η τιμή 32780. Μετά από την εκτέλεση της παραπάνω μεθόδου, περάσαμε την τιμή που θέλαμε από την Java στην R και τώρα μπορούμε να την καλέσουμε και να την χρησιμοποιήσουμε στην R. Ας υποθέσουμε ότι χρειάζεται να προσθέσουμε δύο αριθμούς στην R και να το επιστρέψουμε στην Java (συνήθως δεν χρειάζεται να γίνει κάτι τόσο απλό, αλλά το κάνουμε τώρα για χάριν του παραδείγματος).

Θα χρησιμοποιήσουμε τον εξής κώδικα για να κάνουμε μια πρόσθεση δύο αριθμών.

```
int temp = 32780;
code.addInt("tempvalue", temp);

code.addRCode("sum <- 5 + tempvalue");
caller.setRCode(code);

caller.runAndReturnResult("sum");

int result = caller.getParser().getAsIntArray("sum")[0];
```

Με τον παραπάνω κώδικα, προσθέτουμε 2 αριθμούς μεταξύ τους, τον 32780 που πήρε η R από την Java ως μεταβλητή “tempvalue”, με τον αριθμό 5. Το αποτέλεσμα αυτής της πράξης θα εισαχθεί στην μεταβλητή “sum”, που είναι μεταβλητή της R. Με την χρήση της μεθόδου runAndReturnResult(), που παίρνει ως παράμετρο μια μεταβλητή της R, μπορούμε να εκτελέσουμε την πράξη και να μας επιστρέψει την τιμή της. Έπειτα, με την χρήση της μεθόδου getParser() της κλάσης RCaller, μπορούμε να αναλύσουμε αυτό που θα επιστρέψει. Στην συγκεκριμένη περίπτωση είναι ένας ακέραιος αριθμός (int), οπότε χρησιμοποιούμε την μέθοδο getAsIntArray() που δέχεται ως παράμετρο μια μεταβλητή

της R. Όμως, επειδή αυτή η μέθοδος επιστρέφει έναν πίνακα με τις τιμές που έχει αυτή η μεταβλητή, θα πρέπει να πάρουμε μόνο το πρώτο στοιχείο του πίνακα, οπότε γράφουμε δίπλα από την μέθοδο [0].

Επιπλέον, η RCode παρέχει μεθόδους για μεταφορά πινάκων, καθώς και αντικειμένων:

- `addLogicalArray (String, boolean[])`

```
boolean[] temp = new boolean[]{false, true, false, false, false};
code.addLogicalArray("tempvalue", temp);
```

- `addDoubleArray (String, double[])`

```
double[] temp = new double[]{5.25, 3.14, 2.55, 1.71, 3.43};
code.addDoubleArray("tempvalues", temp);
```

- `addFloatArray (String, float[])`

```
float[] temp = new float[]{3.43f, 4.21f, 5.32f, 7.532f, 5.63f};
code.addFloatArray("tempvalues", temp);
```

- `addShortArray (String, short[])`

```
short[] temp = new short[]{1, 2, 7, 8, 10};
code.addShortArray("tempvalues", temp);
```

- `addIntArray (String, int[])`

```
int[] temp = new int[]{54, 37, 243, 164, 375};
code.addIntArray("tempvalues", temp);
```

- `addLongArray (String, long[])`

```
long[] temp = new long[]{32232, 3123, 2531, 1321, 35367};
code.addLongArray("tempvalues", temp);
```

- `addStringArray (String, String[])`

```
String[] temp = new String[]{"dog", "bee", "cat", "wolf", "ant"};
code.addStringArray("tempvalues", temp);
```

Για να μεταφέρουμε αντικείμενο της Java στην R θα πρέπει να κάνουμε `import` την βιβλιοθήκη «`com.github.rcaller.JavaObject`». Έστω πως έχουμε την εξής κλάση στην Java.

```
public class Point {
    private double x = 5.3;
    private double y = 6.7;
}
```

Με τον παρακάτω κώδικα μπορούμε να περάσουμε ένα αντικείμενο της Java στην R και να αναφερθούμε στις μεταβλητές αυτού του αντικειμένου.

```
code.addJavaObject(new JavaObject("pointobject", new Point()));

code.addRCode("pointobject$x <- 7.8");
code.addRCode("pointobject$y <- 9.3");
caller.setRCode(code);

caller.runAndReturnResult("pointobject");
```


Παράδειγμα χρήσης κάποιας από τις προηγούμενες μεθόδους θα γίνει στην [επόμενη ενότητα](#).

5.3 Τρόποι εκτέλεσης κώδικα στην R

Υπάρχουν τρεις τρόποι για να εκτελέσουμε κώδικα στην R μέσω του RCaller:

- **runOnly()**

Δημιουργεί μια διεργασία Rscript, η οποία δημιουργεί γραφικά.

Συνήθως, χρησιμοποιείται για την εμφάνιση διαφόρων plots (διαγραμμάτων).

```
RCaller caller = RCaller.create();
RCode code = RCode.create();

double[] numbersX = new double[]{2, 3.2, 5.6, 4.3, 9.85, 7.23};
double[] numbersY = new double[]{3.5, 2.3, -3.2, -1.34, 7.65, 5};

code.addDoubleArray("x", numbersX);
code.addDoubleArray("y", numbersY);

File file = code.startPlot();

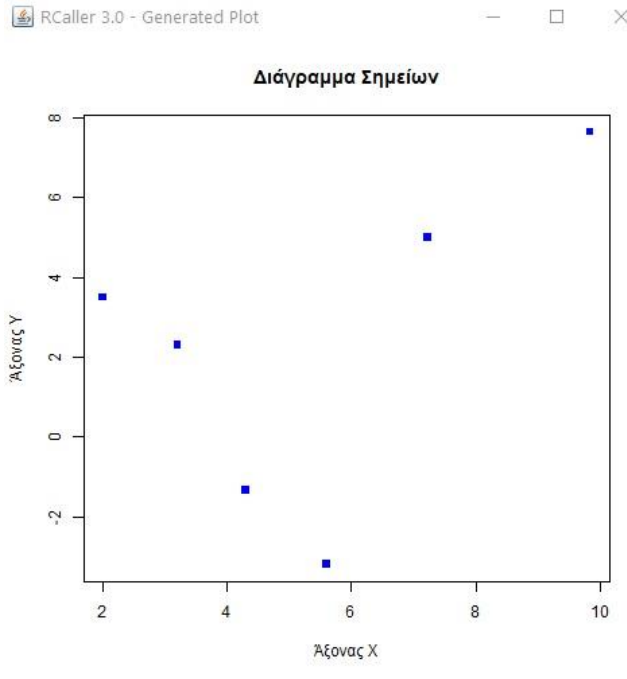
code.addRCode("plot(x, y, xlab='Άξονας X', ylab='Άξονας Y',
main='Διάγραμμα Σημείων', pch=15, col='blue')");

code.endPlot();
caller.setRCode(code);

caller.runOnly();
code.showPlot(file);
```

Οι πρώτες δύο γραμμές του κώδικα χρησιμοποιούνται για την δημιουργία των αντικειμένων της RCaller και RCode, αντίστοιχα. Στις επόμενες δύο γραμμές γεμίζουμε δύο πίνακες double με στοιχεία, όπου ο πίνακας numbersX θα έχει την τετμημένη των σημείων και ο πίνακας numbersY την τεταγμένη των σημείων. Στις επόμενες δύο γραμμές μεταφέρουμε τα στοιχεία των δύο πινάκων σε μεταβλητές της R. Η έβδομη γραμμή του κώδικα χρησιμοποιείται για να ξεκινήσει η δημιουργία μιας γραφικής παράστασης (plot). Η χρήση της εντολής plot χρησιμοποιείται για την δημιουργία μιας γραφικής παράστασης των σημείων x, y και η επόμενη γραμμή του κώδικα για να τερματίσει η δημιουργία της. Με την χρήση της εντολής runOnly(), όπως προαναφέρθηκε μπορούμε να τρέξουμε τον κώδικα της R, χωρίς να περιμένουμε κάποιο αποτέλεσμα στην Java. Έπειτα, εμφανίζουμε την γραφική παράσταση με την εντολή showPlot(), που παίρνει ως

παράμετρο ένα αντικείμενου τύπου File. Το αποτέλεσμα της γραφικής παράστασης εμφανίζεται στην παρακάτω εικόνα.



- **runAndReturnResult()**

Κάθε φορά που εκτελείται αυτή η μέθοδος δημιουργείται μια διεργασία Rscript, η οποία δέχεται ως παράμετρο μια μεταβλητή της R, την οποία και θα επιστρέψει στην Java. Όμως, κάθε φορά που εκτελείται αυτή η μέθοδος, δημιουργείται και μια νέα διεργασία. Έτσι, δεν αποτελεί αποδοτική λύση για επιστροφή πολλαπλών αποτελεσμάτων από διάφορες στατιστικές συναρτήσεις που εκτελέστηκαν από την R.

```
RCaller caller = RCaller.create();
RCode code = RCode.create();

code.addDoubleArray("x", new double[]{2.3, 3.64, -4.45, 4.3, -10});
code.addRCode("result <- mean(x)");

caller.setRCode(code);
caller.runAndReturnResult("result");
double mean = caller.getParser().getAsDoubleArray("result")[0];

System.out.println("mean: " + mean);
```

Με τις πρώτες δύο γραμμές του κώδικα, δημιουργούμε δύο αντικείμενα των κλάσεων RCaller και RCode. Με την τρίτη γραμμή εισάγουμε στην R έναν πίνακα από πέντε αριθμητικές τιμές του πραγματικού συνόλου. Στην τέταρτη γραμμή, με την χρήση της συνάρτησης mean() που παίρνει ως παράμετρο έναν πίνακα από αριθμούς, βρίσκουμε τον μέσο όρο των προηγούμενων αριθμών.

Με τις επόμενες δύο γραμμές στέλνουμε τον κώδικα στην R, τον εκτελούμε και επιστρέφουμε το αποτέλεσμα που έχει αποθηκευτεί στην μεταβλητή “result” της R. Στην έβδομη σειρά του κώδικα περνάμε το πρώτο στοιχείο του πίνακα “result” σε μια μεταβλητή της Java. Στην όγδοη εμφανίζουμε το αποτέλεσμα της πράξης που έγινε στην R, δηλαδή “mean: -0.842”.

- **runAndReturnResultOnline()**

Δημιουργεί μια διεργασία Rscript την οποία κρατάει ανοιχτή, χωρίς να την τερματίσει και δέχεται ως παράμετρο μια μεταβλητή της R. Έτσι, σε αντίθεση με την προηγούμενη μέθοδο, δεν χρειάζεται να δημιουργεί ξανά την διεργασία κάθε φορά που θέλουμε να επιστρέψουμε κάτι από την R στην Java. Όμως, όταν πλέον δε θα θέλουμε να χρησιμοποιήσουμε την διεργασία του Rscript για να εκτελέσουμε κάτι άλλο, θα πρέπει να την τερματίσουμε. Για αυτόν τον λόγο χρησιμοποιείται η μέθοδος stopStreamConsumers() της κλάσης RCaller.

```
RCaller caller = RCaller.create();
RCode code = RCode.create();

code.addDouble("x", -20);
caller.setRCode(code);

code.addRCode("result <- exp(x)");
caller.runAndReturnResultOnline("result");

double exp = caller.getParser().getAsDoubleArray("result")[0];
System.out.println("Εκθετικός αριθμός (e^x): " + exp);

code.clearOnline();

code.addRCode("result <- abs(x)");
caller.runAndReturnResultOnline("result");

double abs = caller.getParser().getAsDoubleArray("result")[0];
System.out.println("Απόλυτος αριθμός: " + abs);

caller.stopStreamConsumers();
```

Με τις πρώτες δύο γραμμές δημιουργούμε δύο αντικείμενα των κλάσεων RCaller και RCode, όπως και στις προηγούμενες περιπτώσεις. Με τις επόμενες δύο γραμμές περνάμε την τιμή -20 στην μεταβλητή x της R και αντιστοιχούμε το αντικείμενο code στο αντικείμενο caller. Στις επόμενες δύο γραμμές βρίσκουμε τον εκθετικό αριθμό του -20 και τον επιστρέφουμε χωρίς να τερματίσουμε τη διεργασία της R που εκτελέσαμε. Μετά περνάμε την τιμή του αποτελέσματος στην μεταβλητή exp της Java και την εμφανίζουμε. Η μέθοδος clearOnline() χρησιμοποιείται για να σβήσουμε όποιον κώδικα του αντικειμένου code δεν έχει

εκτελεστεί ακόμη. Έπειτα, βρίσκουμε τον απόλυτο αριθμό της μεταβλητής x και κάνουμε ότι κάναμε και με τον εκθετικό αριθμό. Επιστρέφουμε την τιμή της πράξης στην μεταβλητή `abs` της Java και μετά την εμφανίζουμε. Μετά εκτελούμε την μέθοδο `stopStreamConsumers()` της κλάσης `RCaller` για να τερματιστεί η διεργασία. Το αποτέλεσμα της εκτέλεσης αυτού του παραδείγματος είναι το εξής:

Εκθετικός αριθμός (e^x): 2.06115362243856E-9
Απόλυτος αριθμός: 20.0

6 Ανάπτυξη εφαρμογής

Η εργασία αποτελείται από πέντε αρχεία .java που βοηθάνε στην εύρεση στοιχείων για τα συστατικά αντικείμενα που έχει κάποιο project, καθώς και στην εύρεση των συσχετίσεων που έχουν μεταξύ τους. Επίσης, υπάρχει και ένα επιπλέον ξεχωριστό αρχείο .java για την εμφάνιση του διαγράμματος της ιεραρχικής συσταδοποίησης.

6.1 Ο πυρήνας της εφαρμογής

Το κύριο μέρος της εφαρμογής αποτελείται από πέντε αρχεία .java, όπως προαναφέρθηκε. Τα δύο εξ' αυτών είναι κλάσεις στις οποίες αποθηκεύονται προσωρινά οι πληροφορίες που βρήκαμε για τα συστατικά του project (MethodCA, CallerCA). Η κλάση MethodDeclarationFinder επιτυγχάνει την μετάβαση στους κόμβους του AST και δημιουργεί μια λίστα με τις μεθόδους που βρέθηκαν. Η κλάση HierarchyCallHelper βρίσκει τις συσχετίσεις που έχουν οι κλάσεις μεταξύ τους και πιο συγκεκριμένα τις μεθόδους που καλούνται από άλλες μεθόδους. Εν τέλει, έχουμε την κλάση GetInfo στην οποία γίνονται αρκετές διεργασίες:

- Βρίσκει τα πακέτα του project που θέλουμε να αναλύσει.
- Δημιουργεί ένα AST δέντρο για κάθε πακέτο του project.
- Δημιουργεί μια λίστα από αντικείμενα MethodCA και αποθηκεύει στις μεταβλητές τους τα συστατικά στοιχεία των μεθόδων που βρίσκει στην ανάλυση.
- Έπειτα δημιουργεί μια λίστα από αντικείμενα CallerCA.
- Και τέλος δημιουργεί το .txt αρχείο με τον πίνακα ομοιότητας των πακέτων και το αποθηκεύει στο home path του χρήστη. Αυτό το αρχείο γίνεται είσοδος στην ιεραρχική συσταδοποίηση που πραγματοποιείται από το έκτο ξεχωριστό αρχείο .java.

Το plugin.xml αποτελεί τον τρόπο δομής του plugin, δηλαδή των commands που το απαρτίζουν (menu, toolbar) και είναι το εξής:

```

<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.4"?>
<plugin>

  <extension
    point="org.eclipse.ui.commands">
    <category
      id="gr.teilar.codeanalyser.commands.category"
      name="Java Code Analyser">
    </category>
    <command
      categoryId="gr.teilar.codeanalyser.commands.category"
      name="Analysis"
      id="gr.teilar.codeanalyser.commands.jcAnalysisCommand">
    </command>
  </extension>
  <extension
    point="org.eclipse.ui.handlers">
    <handler
      class="gr.teilar.codeanalyser.handlers.GetInfo"
      commandId="gr.teilar.codeanalyser.commands.jcAnalysisCommand">
    </handler>
  </extension>
  <extension
    point="org.eclipse.ui.bindings">
    <key
      commandId="gr.teilar.codeanalyser.commands.jcAnalysisCommand"
      schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"
      contextId="org.eclipse.ui.contexts.window"
      sequence="M1+6">
    </key>
  </extension>
  <extension
    point="org.eclipse.ui.menus">
    <menuContribution
      locationURI="toolbar:org.eclipse.ui.main.toolbar?after=additions">
    <toolbar
      id="gr.teilar.codeanalyser.toolbars.sampleToolbar">
    <command
      id="gr.teilar.codeanalyser.toolbars.jcAnalysisCommand"
      commandId="gr.teilar.codeanalyser.commands.jcAnalysisCommand"
      icon="icons/sample.png"
      tooltip="Java Code Analysis">
    </command>
    </toolbar>
    </menuContribution>
  </extension>
</plugin>

```

6.1.1 MethodCA

Η κλάση MethodCA περιέχει μια σειρά από πληροφορίες, οι οποίες προσδιορίζουν μια μέθοδο, καθώς και μεθόδους για την σύγκριση μεταξύ δυο αντικειμένων MethodCA (equals, hashCode) και την εμφάνιση των πληροφοριών της μεθόδου (toString). Οι πληροφορίες πιο συγκεκριμένα είναι το όνομα της κλάσης και του πακέτου στα οποία βρίσκεται η μέθοδος, το όνομα της, μια λίστα με τους τύπους των παραμέτρων που δέχεται, τον τύπο των δεδομένων τον οποίο επιστρέφει η μέθοδος, καθώς και το αναγνωριστικό αυτής (public, private, protected).

```
package gr.teilar.codeanalyser.handlers;

import java.util.ArrayList;
import java.util.List;

public class MethodCA {
    private String packageCA;
    private String classCA;
    private String name;
    private String identifier;
    private String returnType;
    private List<String> parameters = new ArrayList<>();

    public MethodCA() { }

    public MethodCA(String packageCA, String classCA, String name, String identifier,
        String returnType, List<String> parameters) {
        this.packageCA = packageCA;
        this.classCA = classCA;
        this.name = name;
        this.identifier = identifier;
        this.returnType = returnType;
        this.parameters = parameters;
    }

    public String getPackageCA() {
        return packageCA;
    }

    public void setPackageCA(String packageCA) {
        this.packageCA = packageCA;
    }

    public String getClassCA() {
        return classCA;
    }

    public void setClassCA(String classCA) {
        this.classCA = classCA;
    }

    public String getName() {
        return name;
    }
}
```

```

public void setName(String name) {
    this.name = name;
}

public String getIdentifier() {
    return identifier;
}

public void setIdentifier(String identifier) {
    this.identifier = identifier;
}

public String getReturnType() {
    return returnType;
}

public void setReturnType(String returnType) {
    this.returnType = returnType;
}

public List<String> getParameters() {
    return parameters;
}

public void addParameter(String parameter) {
    parameters.add(parameter);
}

public void setParameters(List<String> parameters) {
    this.parameters = parameters;
}

@Override
public String toString() {
    return "MethodCA [packageCA=" + packageCA + ", classCA=" + classCA +
        ", name=" + name + ", identifier=" + identifier +
        ", returnType=" + returnType + ", parameters=" + parameters + "];"
}

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((classCA == null) ? 0 : classCA.hashCode());
    result = prime * result + ((identifier == null) ? 0 : identifier.hashCode());
    result = prime * result + ((name == null) ? 0 : name.hashCode());
    result = prime * result + ((packageCA == null) ? 0 : packageCA.hashCode());
    result = prime * result + ((parameters == null) ? 0 : parameters.hashCode());
    result = prime * result + ((returnType == null) ? 0 : returnType.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;

```



```

        if (getClass() != obj.getClass())
            return false;
        MethodCA other = (MethodCA) obj;
        if (classCA == null) {
            if (other.classCA != null)
                return false;
        } else if (!classCA.equals(other.classCA))
            return false;
        if (identifier == null) {
            if (other.identifier != null)
                return false;
        } else if (!identifier.equals(other.identifier))
            return false;

        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        if (packageCA == null) {
            if (other.packageCA != null)
                return false;
        } else if (!packageCA.equals(other.packageCA))
            return false;
        if (parameters == null) {
            if (other.parameters != null)
                return false;
        } else if (!parameters.equals(other.parameters))
            return false;
        if (returnType == null) {
            if (other.returnType != null)
                return false;
        } else if (!returnType.equals(other.returnType))
            return false;
        return true;
    }
}

```

6.1.2 CallerCA

Η κλάση CallerCA περιέχει δύο μεταβλητές τύπου MethodCA, οι οποίες δείχνουν την συσχέτιση μεταξύ αυτών των δύο, δηλαδή ποια μέθοδος καλείται από ποια. Όπως και στην MethodCA, έτσι και εδώ υπάρχουν μέθοδοι που βοηθούν στην σύγκριση δύο αντικειμένων τύπου CallerCA (equals, hashCode), καθώς και η μέθοδος εμφάνισης αυτών των πληροφοριών (toString).

```

package gr.teilar.codeanalyser.handlers;

public class CallerCA {
    private MethodCA caller;
    private MethodCA callee;

    public CallerCA(MethodCA caller, MethodCA callee) {
        this.caller = caller;
        this.callee = callee;
    }
}

```

```

public MethodCA getCaller() {
    return caller;
}

public void setCaller(MethodCA caller) {
    this.caller = caller;
}

public MethodCA getCallee() {
    return callee;
}

public void setCallee(MethodCA callee) {
    this.callee = callee;
}

@Override
public String toString() {
    return "CallerCA [caller=" + caller + ", callee=" + callee + "]";
}

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((callee == null) ? 0 : callee.hashCode());
    result = prime * result + ((caller == null) ? 0 : caller.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    CallerCA other = (CallerCA) obj;
    if (callee == null) {
        if (other.callee != null)
            return false;
    } else if (!callee.equals(other.callee))
        return false;
    if (caller == null) {
        if (other.caller != null)
            return false;
    } else if (!caller.equals(other.caller))
        return false;
    return true;
}
}

```

6.1.3 MethodDeclarationFinder

Η κλάση MethodDeclarationFinder περιέχει μία λίστα methods από αντικείμενα MethodDeclaration και είναι οι μέθοδοι τις οποίες επισκέπτεται το μοντέλο AST με την

χρήση της μεθόδου visit. Η μέθοδος getMethods επιστρέφει την λίστα των μεθόδων με τέτοιο τρόπο, ώστε να μην μπορούν να πειραχτούν οι εγγραφές της λίστας.

```
package gr.teilar.codeanalyser.handlers;

import java.util.ArrayList;
import java.util.List;
import java.util.Collections;

import org.eclipse.jdt.core.dom.ASTVisitor;
import org.eclipse.jdt.core.dom.ASTNode;
import org.eclipse.jdt.core.dom.MethodDeclaration;

public final class MethodDeclarationFinder extends ASTVisitor {
    private final List<MethodDeclaration> methods = new ArrayList<>();

    @Override
    public boolean visit(final MethodDeclaration method) {
        methods.add(method);
        return super.visit(method);
    }

    public List<MethodDeclaration> getMethods() {
        return Collections.unmodifiableList(methods);
    }
}
```

6.1.4 HierarchyCallHelper

Η κλάση HierarchyCallHelper περιέχει ένα αντικείμενο τύπου CallHierarchy, έναν κατασκευαστή χωρίς ορίσματα, στον οποίο παίρνει τιμή το αντικείμενο callHierarchy της κλάσης και ενεργοποιούμε τα φίλτρα με την χρήση της μεθόδου setFilterEnabled(true), έπειτα χρησιμοποιείται η μέθοδος setFilters(), στην οποία βάζουμε ως όρισμα ένα String με πακέτα τα οποία δε θέλουμε να συμπεριλάβει στην εύρεση των συσχετίσεων. Επίσης, υπάρχει η μέθοδος getCallHierarchy() που επιστρέφει το αντικείμενο callHierarchy. Ο μόνος λόγος που χρειαζόμαστε αυτή την μέθοδο είναι για να απενεργοποιήσουμε τα φίλτρα, όταν τελειώσει η εύρεση των συσχετίσεων. Επειδή χρησιμοποιούνται εσωτερικές κλάσεις του eclipse πρέπει να γίνει η απενεργοποίηση των φίλτρων για να μην αποτελεί πρόβλημα στην λειτουργία της ιεραρχίας που έχει το eclipse. Η μέθοδος getCallersOf() δέχεται ως όρισμα ένα αντικείμενο τύπου IMethod και επιστρέφει ένα HashSet με τις μεθόδους οι οποίες καλούν την μέθοδο που ορίσαμε. Η getIMethods() μας βοηθάει να βρούμε τα αντικείμενα IMethod που βρίσκονται στον πίνακα με τα MethodWrapper. Τώρα, αν κάποιο συγκεκριμένο MethodWrapper από τον προαναφερθέντα πίνακα δεν είναι IMethod, θα καλεστεί η μέθοδος getIMethodFromMethodWrapper και θα επιστραφεί η μέθοδος, εάν αυτή υπάρχει και είναι τύπου IMethod.

```

package gr.teilar.codeanalyser.handlers;

import java.util.HashSet;

import org.eclipse.core.runtime.NullProgressMonitor;
import org.eclipse.jdt.core.IJavaElement;
import org.eclipse.jdt.core.IMember;
import org.eclipse.jdt.core.IMethod;
import org.eclipse.jdt.internal.corext.callhierarchy.CallHierarchy;
import org.eclipse.jdt.internal.corext.callhierarchy.MethodWrapper;

public class HierarchyCallHelper {
    private CallHierarchy callHierarchy;

    public HierarchyCallHelper() {
        callHierarchy = CallHierarchy.getDefault();
        callHierarchy.setFilterEnabled(true);
        callHierarchy.setFilters("java.*", "javax.*", "javafx.*", "com.sun.*");
    }

    public CallHierarchy getCallHierarchy() {
        return callHierarchy;
    }

    public HashSet<IMethod> getCallersOf(IMethod m) {
        IMember[] members = { m };

        MethodWrapper[] methodWrappers = callHierarchy.getCallerRoots(members);
        HashSet<IMethod> callers = new HashSet<IMethod>();
        for (MethodWrapper mw : methodWrappers) {
            MethodWrapper[] mw2 = mw.getCalls(new NullProgressMonitor());
            HashSet<IMethod> temp = getIMethods(mw2);
            callers.addAll(temp);
        }

        return callers;
    }

    HashSet<IMethod> getIMethods(MethodWrapper[] methodWrappers) {
        HashSet<IMethod> c = new HashSet<IMethod>();
        for (MethodWrapper m : methodWrappers) {
            IMethod im = getIMethodFromMethodWrapper(m);
            if (im != null) {
                c.add(im);
            }
        }
        return c;
    }

    IMethod getIMethodFromMethodWrapper(MethodWrapper m) {
        try {
            IMember im = m.getMember();
            if (im.getElementType() == IJavaElement.METHOD) {
                return (IMethod) m.getMember();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}

```

```
}  
}
```

6.1.5 GetInfo

Η κλάση GetInfo περιέχει τον πυρήνα όλης της εργασίας και ευθύνεται για την δημιουργία του plugin, την εύρεση των συσχετίσεων, καθώς και την έξοδο των συσχετίσεων σε αρχείο με τέτοιο τρόπο που να είναι χρήσιμος για την εμφάνιση των δεδομένων σε ιεραρχικό διάγραμμα. Η κλάση περιέχει μια λίστα methodsCA από αντικείμενα τύπου MethodCA που είναι οι μέθοδοι που περιέχει το project που θέλουμε να αναλύσουμε, καθώς και μια λίστα callersCA από αντικείμενα τύπου CallerCA που είναι οι συσχετίσεις μεταξύ των μεθόδων. Επιπλέον περιέχει και ένα αντικείμενο τύπου HierarchyCallHelper που όπως προαναφέρθηκε βοηθάει στην εύρεση των συσχετίσεων.

Η μέθοδος execute() καλείται, όταν εκτελεστεί το command του plugin, δηλαδή όταν πατηθεί το εικονίδιο του plugin που βρίσκεται στο toolbar. Έτσι, όταν εκτελεστεί η μέθοδος θα ψάξει για τα project που βρίσκονται στο IDE του eclipse, εάν αυτά τα project είναι ανοιχτά και είναι project τύπου java (δηλαδή είναι project που έχουν ενεργοποιημένο το «org.eclipse.jdt.core.javanature»), τότε η μέθοδος execute() θα καλέσει την μέθοδο analyseMethods() που έχει ως όρισμα το ίδιο το project που έχει περάσει από τις συνθήκες που προαναφέρθηκαν.

Η μέθοδος analyseMethods() έχει ως όρισμα ένα αντικείμενο τύπου IProject και δημιουργεί τα arraylists callersCA και methodsCA από την αρχή, ώστε να μην περιέχουν ήδη τιμές από κάποια ανάλυση προηγούμενου project. Έπειτα με την χρήση της μεθόδου getPackageFragments(), παίρνουμε τα πακέτα που έχει το project και προχωράμε στην πρώτη ανάλυση που είναι η εύρεση των μεθόδων που έχουν τα πακέτα με την χρήση της μεθόδου createAST(mypackage, 0), εφόσον τελειώσει με αυτήν θα πάει στην δεύτερη ανάλυση createAST(mypackage, 1) για να βρει τις συσχετίσεις μεταξύ αυτών των μεθόδων. Ύστερα, με την χρήση της μεθόδου createOutput() που έχει ως ορίσματα την λίστα callersCA, καθώς και την methodsCA φέρνει σε κατάλληλη μορφή τα δεδομένα των συσχετίσεων με βάση κάποια απλοϊκή μετρική ($\frac{\text{αριθμός των κλάσεων που καλούνται από ένα πακέτο}}{\text{αριθμός των κλάσεων που έχει το πακέτο}}$) και βγάζει ως έξοδο ένα αρχείο .txt με αυτά τα δεδομένα, έπειτα εφόσον έχει τελειώσει η ανάλυση με την χρήση της

μεθόδου `getCallHierarchy()` παίρνουμε το αντικείμενο `CallHierarchy` που δημιουργήθηκε και απενεργοποιούμε τα φίλτρα που χρησιμοποιήθηκαν για την εύρεση των συσχετίσεων.

Η μετρική $\frac{\text{αριθμός των κλάσεων που καλούνται από ένα πακέτο}}{\text{αριθμός των κλάσεων που έχει το πακέτο}}$ επιστρέφει τιμές που βρίσκονται στο διάστημα $[0.0, 1.0]$. Η κύρια διαγώνιος του πίνακα ομοιότητας παίρνει μηδενικές τιμές, γιατί δεν μπορούμε να εξάγουμε πληροφορίες για συσχετίσεις όταν το πακέτο που καλεί τις κλάσεις είναι το ίδιο το πακέτο που έχει τις κλάσεις. Όταν οι τιμές των κελιών του πίνακα ομοιότητας μεταξύ δύο πακέτων είναι μεγάλες τότε αυτά τα δύο πακέτα θα συγχωνευτούν μεταξύ τους σε κάποιο module, γιατί γίνεται συχνή χρήση των κλάσεων του ενός πακέτου, στο άλλο πακέτο. Οπότε οι λόγοι που χρησιμοποιήθηκε αυτή η μετρική, είναι κυρίως η απλότητά της, καθώς και ο λόγος μεταξύ αυτών των δύο οντοτήτων, ο οποίος μας δίνει πραγματικές τιμές μεταξύ 0 και 1.

Πιο συγκεκριμένα, η μέθοδος `createAST()` δέχεται ως ορίσματα ένα αντικείμενο τύπου `IPackageFragment` και έναν ακέραιο αριθμό. Όταν αυτός ο ακέραιος αριθμός είναι 0 δημιουργείται ένα αντικείμενο `MethodCA` στο οποίο αποθηκεύονται οι πληροφορίες της μεθόδου που αναλύεται και προστίθεται στην λίστα `methodsCA`. Εάν αυτός ο ακέραιος αριθμός έχει οποιαδήποτε άλλη τιμή τότε προχωράμε στην εύρεση των συσχετίσεων για κάθε μέθοδο που αναλύεται με την χρήση του `HashSet` που επιστρέφει η μέθοδος `getCallersOf()`. Έπειτα, εάν υπάρχουν και οι δύο μέθοδοι στην λίστα `methodsCA` (η πρώτη μέθοδος είναι αυτή που καλεί την άλλη μέθοδο (caller), και η δεύτερη αυτή που καλείται από την πρώτη (callee)), τότε δημιουργείται ένα αντικείμενο τύπου `CallerCA` και προστίθεται στην λίστα `callersCA`.

Η μέθοδος `getMDNode()` παίρνει ως όρισμα ένα αντικείμενο τύπου `IMethod` και επιστρέφει το αντίστοιχο `MethodDeclaration` (υποκατηγορία ενός `ASTNode`) αντικείμενο για αυτήν με την χρήση του `NodeFinder`. Αυτή η μέθοδος χρησιμοποιείται, γιατί θέλουμε να πάρουμε επιπλέον στοιχεία για τις παραμέτρους που έχει μια μέθοδος, τις οποίες δε θα μπορούσαμε να πάρουμε αν ήταν `IMethod`.

Η μέθοδος `getMethodFromList()` μετά από μια σειρά ελέγχων με βάση τις παραμέτρους, τον τύπο δεδομένων που επιστρέφει καθώς και το `MethodDeclaration` που έχει μια μέθοδος, επιστρέφει ένα αντικείμενο `MethodCA` που βρίσκεται στην λίστα `methodsCA`, εάν αυτό υπάρχει φυσικά. Η μέθοδος `parse()` χρησιμοποιείται για την δημιουργία ενός

AST και έχει παίρνει ως παράμετρο ένα αντικείμενο τύπου ICompilationUnit και επιστρέφει ένα αντικείμενο τύπου CompilationUnit, το οποίο είναι ένα ASTNode που περιέχει την ρίζα ενός AST.

```
package gr.teilar.codeanalyser.handlers;

import java.io.IOException;

import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.eclipse.core.commands.AbstractHandler;
import org.eclipse.core.commands.ExecutionEvent;
import org.eclipse.core.commands.ExecutionException;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IWorkspaceRoot;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.core.runtime.CoreException;

import org.eclipse.jdt.core.ICompilationUnit;
import org.eclipse.jdt.core.IMethod;
import org.eclipse.jdt.core.IPackageFragment;
import org.eclipse.jdt.core.IPackageFragmentRoot;
import org.eclipse.jdt.core.JavaCore;
import org.eclipse.jdt.core.JavaModelException;
import org.eclipse.jdt.core.dom.AST;
import org.eclipse.jdt.core.dom.ASTNode;
import org.eclipse.jdt.core.dom.ASTParser;
import org.eclipse.jdt.core.dom.CompilationUnit;
import org.eclipse.jdt.core.dom.IMethodBinding;
import org.eclipse.jdt.core.dom.ITypeBinding;
import org.eclipse.jdt.core.dom.MethodDeclaration;
import org.eclipse.jdt.core.dom.Modifier;
import org.eclipse.jdt.core.dom.NodeFinder;

public class GetInfo extends AbstractHandler {
    private static List<MethodCA> methodsCA;
    private static List<CallerCA> callersCA;
    private static final String JDT_NATURE = "org.eclipse.jdt.core.javanature";
    HierarchyCallHelper hc = new HierarchyCallHelper();

    @Override
    public Object execute(ExecutionEvent event) throws ExecutionException {
        IWorkspaceRoot root = ResourcesPlugin.getWorkspace().getRoot();

        // Get all projects in the workspace
        IProject[] projects = root.getProjects();
        // Loop over all projects
        for (IProject project : projects) {
            if(project.isOpen()) {
                try {
                    if (project.isNatureEnabled(JDT_NATURE)) {
```

```

        System.out.println("Analyzing project \"" + project.getName() + "\".");
        analyseMethods(project);
    }
} catch (CoreException e) {
    e.printStackTrace();
}
}
else
    System.out.println("\"" + project.getName() + "\" is a closed project.");
}
return null;
}

private void analyseMethods(IProject project) throws JavaModelException {
    // reset arraylists if analyser is pressed again
    methodsCA = new ArrayList<MethodCA>();
    callersCA = new ArrayList<CallerCA>();
    IPackageFragment[] packages = JavaCore.create(project).getPackageFragments();
    // parse(JavaCore.create(project));
    for (IPackageFragment mypackage : packages) {
        if (mypackage.getKind() == IPackageFragmentRoot.K_SOURCE) {
            createAST(mypackage, 0);
        }
    }

    methodsCA.forEach((MethodCA m) -> {
        System.out.println(m.toString());
    });
    System.out.println("=====
=====");

    for (IPackageFragment mypackage : packages) {
        if (mypackage.getKind() == IPackageFragmentRoot.K_SOURCE) {
            createAST(mypackage, 1);
        }
    }

    callersCA.forEach((CallerCA m) -> {
        System.out.println(m.toString());
    });
    System.out.println("=====
=====");

    createOutput(callersCA, methodsCA);

    hc.getCallHierarchy().setFilterEnabled(false);
}

private void createAST(IPackageFragment mypackage, int usage) throws JavaModelException {
    for (ICompilationUnit unit : mypackage.getCompilationUnits()) {
        // now create the AST for the ICompilationUnits
        CompilationUnit parse = parse(unit);
        MethodDeclarationFinder visitor = new MethodDeclarationFinder();

        parse.accept(visitor);

        for (MethodDeclaration method : visitor.getMethods()) {
            IMethodBinding mBinding = method.resolveBinding();

```



```

List<String> paramsCallee = new ArrayList<>();
ITypeBinding[] bindCallee = mBinding.getParameterTypes();
String returnTypeCallee = "";

for (ITypeBinding param : bindCallee)
    paramsCallee.add(param.getName());

int i = method.getModifiers();

String identifierCallee = "";
if (Modifier.isPrivate(i))
    identifierCallee = "private";
if (Modifier.isProtected(i))
    identifierCallee = "protected";
if (Modifier.isPublic(i))
    identifierCallee = "public";

if (method.getReturnType2() != null)
    returnTypeCallee = method.getReturnType2().toString();

if (usage == 0 && !mBinding.getDeclaringClass().getName().toString().equals("")
    && !mBinding.getDeclaringClass().getName().equals("Null")) {
    MethodCA m = new MethodCA(mypackage.getElementName(),
        mBinding.getDeclaringClass().getName().toString(),
method.getName().toString(),
        identifierCallee, returnTypeCallee, paramsCallee);

    methodsCA.add(m);
} else {
    IMethod p = (IMethod) mBinding.getJavaElement();

    for(IMethod o : hc.getCallersOf(p)) {
        MethodDeclaration md = getMDNode(o);
        List<String> paramsCaller = new ArrayList<>();
        ITypeBinding[] bindCaller = md.resolveBinding().getParameterTypes();

        String returnTypeCaller = "";

        for (ITypeBinding param : bindCaller)
            paramsCaller.add(param.getName());

        if (md.getReturnType2() != null)
            returnTypeCaller = md.getReturnType2().toString();

        if (getMethodFromList(methodsCA, md, paramsCaller, returnTypeCaller) != null)
        {
            MethodCA caller = getMethodFromList(methodsCA, md, paramsCaller,
returnTypeCaller);

            if (getMethodFromList(methodsCA, method, paramsCallee, returnTypeCallee)
!= null) {
                MethodCA callee = getMethodFromList(methodsCA, method, paramsCallee,
returnTypeCallee);

                CallerCA call = new CallerCA(caller, callee);

                callersCA.add(call);
            }
        }
    }
}

```

```

    }
    }
}

private void createOutput(List<CallerCA> callers, List<MethodCA> methods) {
    List<String> packageList = new ArrayList<>();
    List<String> remClassList = new ArrayList<>();

    for(CallerCA o : callers) {
        String callerPackage = o.getCaller().getPackageCA();
        String calleePackage = o.getCallee().getPackageCA();

        if(!packageList.contains(callerPackage))
            packageList.add(callerPackage);

        if(!packageList.contains(calleePackage))
            packageList.add(calleePackage);
    }

    int[] packageSize = new int[packageList.size()];

    for(MethodCA p : methods) {
        for(int i = 0; i < packageSize.length; i++) {
            if(p.getPackageCA().equals(packageList.toArray()[i])) {
                if(!remClassList.contains(p.getClassCA())) {
                    packageSize[i]++;
                    remClassList.add(p.getClassCA());
                }
            }
        }
    }
    remClassList.clear();

    String fileContent[] = new String[packageList.size()+1];
    fileContent[0] = "";

    for(int i = 0; i < packageList.toArray().length; i++) {
        fileContent[0] += packageList.toArray()[i].toString();

        if(i != (packageList.toArray().length - 1))
            fileContent[0] += " ";
    }

    System.out.println("Package \tNumber of Classes");
    for(int i=0; i < packageList.toArray().length; i++) {
        System.out.println(packageList.toArray()[i] + "\t" + packageSize[i]);
    }

    double array[][] = new double[packageList.size()][packageList.size()];

    System.out.println("=====");
    System.out.println("Similarity matrix got saved.");
    for(int i = 0; i < array.length; i++) {
        int temp = i+1;
        fileContent[temp] = "";
    }
}

```

```

        for(int j = 0; j < array[i].length; j++) {
            int counter = 0;

            for(CallerCA o : callers) {
                if(!packageList.toArray()[i].equals(packageList.toArray()[j])) {
                    if(o.getCaller().getPackageCA().equals(packageList.toArray()[i]) &&
o.getCallee().getPackageCA().equals(packageList.toArray()[j])) {
                        if(!remClassList.contains(o.getCallee().getClassCA())) {
                            counter++;
                            remClassList.add(o.getCallee().getClassCA());
                        }
                    }
                }
            }

            remClassList.clear();

            if(i != j)
                array[i][j] = (counter / (double) packageSize[j]);

            fileContent[temp] += array[i][j];

            if(j != (array[i].length - 1))
                fileContent[temp] += " ";
        }
    }

    String directory = System.getProperty("user.home");
    Path path = Paths.get(directory, "output.txt");

    List<String> output = Arrays.asList(fileContent);

    try
    {
        Files.write(path, output, StandardOpenOption.CREATE);
    }
    catch(IOException ex)
    {
        ex.printStackTrace();
    }
}

public static MethodDeclaration getMDNode(IMethod method) throws JavaModelException {
    ICompilationUnit methodCompilationUnit = method.getCompilationUnit();
    CompilationUnit unit = parse(methodCompilationUnit); // returns ASTNode

    ASTNode currentNode = NodeFinder.perform(unit, method.getSourceRange());

    return (MethodDeclaration) currentNode;
}

private static MethodCA getMethodFromList(List<MethodCA> m, MethodDeclaration md,
    List<String> mdParams, String mdRetTypeCaller) {

    MethodCA ret = null;
    String mdPackageName = md.getRoot().toString().split("package ")[1].split(";")[0];

    for (MethodCA o : m) {

```

```

        if (o.getName().equals(md.getName().toString()) && o.getReturnType()
.equals(mdRetTypeCaller)) {
            if (o.getParameters().equals(mdParams) && o.getPackageCA().equals(mdPackageName)) {
                if (o.getClassCA().equals(md.resolveBinding().getDeclaringClass()
.getName().toString())) {
                    ret = o;
                    break;
                }
            }
        }
    }
    return ret;
}

private static CompilationUnit parse(CompilationUnit unit) {
    ASTParser parser = ASTParser.newParser(AST.JLS10);
    parser.setKind(ASTParser.K_COMPILATION_UNIT);
    parser.setSource(unit);
    parser.setResolveBindings(true);
    return (CompilationUnit) parser.createAST(null);
}
}

```

6.2 Εμφάνιση του διαγράμματος

Ο παρακάτω κώδικας δεν αποτελεί μέρος του plugin, αλλά είναι ένα ξεχωριστό αρχείο. Με την χρήση της βιβλιοθήκης RCaller επικαλούμαστε τις μεθόδους της, για την επικοινωνία της Java με την R, ώστε να γίνει η εύρεση και η δημιουργία του διαγράμματος. Πρώτα δημιουργούνται δύο αντικείμενα τύπου RCaller και RCode αντίστοιχα. Έπειτα, βάζουμε σε ένα String την τοποθεσία του αρχείου .txt που έβγαλε η ανάλυση και με την χρήση της μεθόδου addString(), τοποθετούμε το προηγούμενο String στην μεταβλητή path της R. Ύστερα, με την χρήση του addRCode() τρέχουμε τον κώδικα για την εύρεση του ιεραρχικού διαγράμματος στην R. Πρώτα, πρέπει να διαβαστεί το αρχείο .txt με την εντολή read.csv() και το αποτέλεσμα αποθηκεύεται στην μεταβλητή input. Αυτή η εντολή έχει ως παραμέτρους το path, δηλαδή το μονοπάτι στο οποίο βρίσκεται το αρχείο, η παράμετρος sep (separator) είναι ο κενός χαρακτήρας, γιατί σε τέτοια μορφή βγάζει την έξοδο το plugin και επειδή η πρώτη γραμμή του αρχείου περιέχει τα ονόματα των πακέτων, βάζουμε στην παράμετρο header την τιμή TRUE για να την παραλείψει. Μετά, στην μεταβλητή dist_mat, με την χρήση της εντολής dist() αποθηκεύονται οι αποστάσεις που έχουν μεταξύ τους οι τιμές των πακέτων και με την χρήση της μεθόδου hclust() δημιουργείται η ιεραρχική συσταδοποίηση των πακέτων με την μέθοδο average. Εν τέλει, δημιουργείται ένα plot που έχει ως colnames, τα ονόματα των πακέτων.

Ο κώδικας εκτελείται και εμφανίζει το διάγραμμα της ιεραρχικής συσταδοποίησης των πακέτων.

```
package gr.teilar.codeanalyser.rcaller;

import java.io.File;
import java.util.logging.Level;
import java.util.logging.Logger;

import com.github.rcaller.rstuff.RCaller;
import com.github.rcaller.rstuff.RCode;

public class Main {
    public static void main(String[] args) {
        try {
            RCaller caller = RCaller.create();
            RCode code = RCode.create();

            String path = System.getProperty("user.home") + "\\output.txt";
            path = path.replace("\\", "\\");

            code.addString("path", path);

            code.addRCode("input <- read.csv(path, sep = ' ', header = TRUE)");
            code.addRCode("dist_mat <- dist(as.matrix(input))");
            code.addRCode("hclust_ave <- hclust(dist_mat, method = 'average')");

            File file = code.startPlot();
            code.addRCode("plot(hclust_ave, labels=colnames(input))");
            code.endPlot();

            caller.setRCode(code);

            caller.runOnly();
            code.showPlot(file);
        } catch (Exception e) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, e.getMessage());
        }
    }
}
```

Ας πάρουμε για παράδειγμα ότι έχουμε την εξής έξοδο από το plugin:

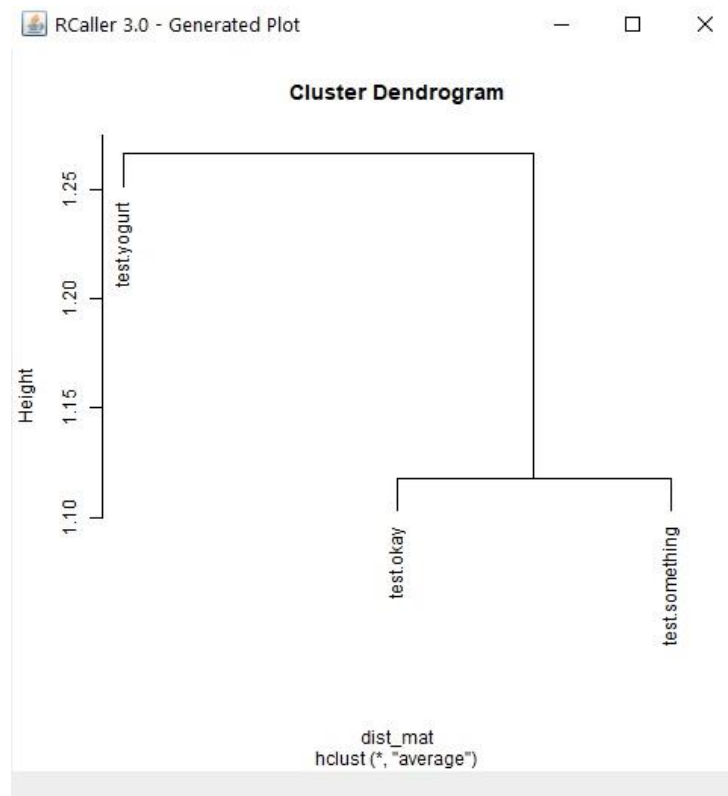
```
test.okay test.something test.yogurt
```

```
0.0 1.0 1.0
```

```
0.5 0.0 1.0
```

```
0.0 0.0 0.0
```

Τότε το διάγραμμα που θα εμφανιστεί θα είναι το εξής:



Αυτό σημαίνει πως τα πακέτα `test.okay` και `test.something` έχουν μεθόδους που καλούν πολύ ή μία την άλλη, οπότε θα ήταν καλύτερο σε κάποιο επόμενο βήμα να συγχωνευτούν σε ένα module, όπως φαίνεται και στο ιεραρχικό διάγραμμα.

7 Συμπεράσματα

Το θέμα της εργασίας ήταν η δημιουργία ενός plugin για το IDE του Eclipse, το οποίο αναλύει στατικά την δομή ενός project, εξάγει τα συμπεράσματα της ανάλυσης σε ένα αρχείο που μετά αποτελεί είσοδο σε μια εφαρμογή που ενώ είναι γραμμένη σε Java εκτελεί κώδικα της R για την δημιουργία της ιεραρχικής συσταδοποίησης των πακέτων. Αυτή η πτυχιακή αποτέλεσε μια αφορμή για σκέψη σχετικά με την δομή που είναι καλό να έχουν τα project της Java και μου έδωσε την ευκαιρία να ασχοληθώ με το στατιστικό πακέτο της R. Το μόνο που αποτέλεσε εμπόδιο στην ομαλή ανάπτυξη του κώδικα ήταν η χρήση της εσωτερικής βιβλιοθήκης «CallHierarchy» του Eclipse, διότι δεν υπήρχε το κατάλληλο documentation για την χρήση των μεθόδων της. Οπότε, αν θα ξαναέκανα την εργασία θα προσπαθούσα να βρω κάποιον άλλον τρόπο για την εύρεση των συσχετίσεων. Η μετρική που χρησιμοποιήθηκε για την εύρεση των συγγενικών μεταξύ τους πακέτων ήταν πολύ απλοϊκή και μια άμεση αναβάθμιση σε αυτήν θα ήταν η χρησιμοποίηση κάποιας μετρικής του Martin. Επιπλέον, θα μπορούσε η εκτέλεση του κώδικα της R να γίνεται από το ίδιο το plugin και όχι από κάποιο ξεχωριστό πρόγραμμα.

Βιβλιογραφία

- [1] Rokach, Lior, and Oded Maimon. "Clustering methods." Data mining and knowledge discovery handbook. Springer US, 2005. 321-352
- [2] vogella and Eclipse JDT, “Abstract Syntax Tree and Java Model”, 2018
- [3] Mehmet Hatan Satman and Paul Curcean, “RCaller Documentation”, 2016