

ΙΟΝΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ «ΨΗΦΙΑΚΕΣ ΕΦΑΡΜΟΓΕΣ &
ΚΑΙΝΟΤΟΜΙΑ»



– Capstone Project –

Η χρήση των συντακτικών δέντρων στην δημιουργία
μιας εφαρμογής ανάλυσης προγραμμάτων Java

Νικολέτα Τρύφωνα
inf.dig2207
Επιβλέπων: – Θεόδωρος Ανδρόνικος –

26 Ιανουαρίου 2024

Επιβλέπων

Θεόδωρος Ανδρόνικος, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο

Τριμελής Επιτροπή

Θεόδωρος Ανδρόνικος, Αναπληρωτής Καθηγητής,
Ιόνιο Πανεπιστήμιο

Ελένη Χριστοπούλου, Εργαστηριακό Διδακτικό Προσωπικό,
Ιόνιο Πανεπιστήμιο

Δημήτριος Ρίγγας, Εργαστηριακό Διδακτικό Προσωπικό,
Ιόνιο Πανεπιστήμιο

Περίληψη

Σκοπός αυτής της διπλωματικής εργασίας είναι η ανάπτυξη ενός λογισμικού το οποίο λειτουργεί ως ένα εργαλείο ανάλυσης προγραμμάτων Java. Αυτό σημαίνει ότι το νέο πρόγραμμα είναι σε θέση να εμφανίζει τα bytecodes που αντιστοιχούν σε ένα αρχείο το οποίο δίνει ως είσοδο ο χρήστης. Το αρχείο εισόδου μπορεί να είναι ένα ήδη μεταγλωττισμένο αρχείο (με κατάληξη .class) ή ένα αρχείο πηγαίου κώδικα το οποίο αναλαμβάνει να μεταγλωττίσει το ίδιο το πρόγραμμα με σκοπό να εμφανίσει τα bytecodes που αντιστοιχούν σε αυτό. Η δεύτερη λειτουργία που επιτελεί το νέο αυτό λογισμικό είναι αυτή του decompiler. Με άλλα λόγια, ο χρήστης μπορεί να δίνει ως είσοδο ένα .class αρχείο και να παίρνει ως έξοδο τον πηγαίο κώδικα που αντιστοιχεί σε αυτό. Αυτές οι δύο λειτουργίες πραγματοποιούνται μέσω μοντέρνων αλγορίθμων οι οποίοι κάνουν χρήση της βιβλιοθήκης ανοιχτού κώδικα javassist.

Πέραν όμως από την ανάπτυξη του προαναφερθέντος λογισμικού, αυτή η πτυχιακή εργασία έχει και άλλους στόχους. Επιπλέον λοιπόν στόχος είναι η ανάλυση και επεξήγηση της λειτουργίας κάθε bytecode της γλώσσας Java. Αυτό το κομμάτι είναι ιδιαίτερα σημαντικό διότι τα bytecodes είναι οι θεμελιώδεις λίθοι της γλώσσας Java. Επίσης, η κατανόηση τους είναι εξαιρετικής σημασίας για την κατανόηση των μοντέρνων αλγορίθμων που χρησιμοποιούνται στο αναπτυχθέν λογισμικό. Συγκεκριμένα, η ανάλυση των bytecodes γίνεται σε σχέση με την αλληλεπίδραση τους με την κύρια δομή δεδομένων που χρησιμοποιείται η οποία είναι η στοίβα τελεστών (operand stack). Η ανάλυση αυτή γίνεται μετά την ολοκλήρωση της εισαγωγής, η οποία περιέχει μια αναφορά στον τρόπο λειτουργίας της γλώσσας Java αλλά και των δομών που αυτή χρησιμοποιεί.

Επόμενος στόχος αυτής της εργασίας είναι η ενημέρωση σχετικά με άλλα υπάρχοντα λογισμικά τα οποία επιτελούν την ίδια εργασία. Συγκεκριμένα, γίνεται αναφορά των λειτουργιών που προσφέρουν, των πλεονεκτημάτων τους και τυχόν μειονεκτημάτων τους. Σε επόμενο κεφάλαιο γίνεται ανάλυση των αλγορίθμων που χρησιμοποιούνται από το λογισμικό. Εκτός από την περιγραφή των αλγορίθμων αναλύεται και βήμα βήμα το πρόγραμμα εξηγώντας τις

δομές δεδομένων που χρησιμοποιούνται καθώς και τα βήματα που εκτελούνται με σκοπό την επίτευξη του τελικού αποτελέσματος.

Στο τελευταίο κεφάλαιο, αναφέρονται τα παραδείγματα. Τα παραδείγματα παρουσιάζονται με την εμφάνιση στιγμιοτύπων λειτουργίας του προγράμματος (εικόνων) ώστε να φαίνεται καθαρά η έξοδος.

Τέλος, στο παράρτημα δίνεται ο πηγαίος κώδικας του αναπτυχθέντος λογισμικού σχολιασμένος καθώς και η βιβλιογραφία.

Περιεχόμενα

A´	ΕΙΣΑΓΩΓΗ	1
A´.1	Η ΙΣΤΟΡΙΑ ΤΗΣ JAVA	1
A´.2	Η ΔΟΜΗ ΤΗΣ ΕΙΚΟΝΙΚΗΣ ΜΗΧΑΝΗΣ ΤΗΣ JAVA	4
A´.2.1	Τύποι δεδομένων	4
A´.2.2	Περιοχές δεδομένων κατά τη διάρκεια εκτέλεσης των προγραμμάτων	6
A´.2.2.1	Ο καταχωρητής pc (program counter)	6
A´.2.2.2	Οι στοίβες της εικονικής μηχανής	6
A´.2.2.3	Ο σωρός (Heap)	7
A´.2.2.4	Η περιοχή των μεθόδων (Method Area)	7
A´.2.2.5	Runtime Constant Pool	8
A´.2.3	Τα πλαίσια (frames)	8
A´.2.3.1	Τοπικές μεταβλητές (Local Variables)	9
A´.2.3.2	Στοιβες Δεδομένων (Operand Stacks)	10
A´.2.3.3	Ειδικές μέθοδοι αρχικοποίησης	10
A´.3	Η ΔΟΜΗ ΚΑΙ Η ΔΙΑΤΑΞΗ ΕΝΟΣ CLASS ΑΡΧΕΙΟΥ	11
A´.3.1	Η δομή του class αρχείου	11
A´.3.2	Descriptors	14
A´.3.2.1	Descriptors πεδίων	14
A´.3.2.2	Descriptors μεθόδων	16
B´	ΣΚΟΠΟΙ ΧΡΗΣΗΣ &ΠΑΡΕΜΦΕΡΗΣ ΕΡΕΤΝΑ	17
B´.0.1	Σκοποί χρήσης των decompilers	17
B´.0.2	Ο decompiler DAVA	19
B´.0.3	Ο decompiler JODE	19

B'.0.4	O decompiler JAD	20
B'.0.5	O decompiler JD	20
B'.0.6	O decompiler Mocha	20
B'.0.7	O decompiler DeJaVu	21
B'.0.8	O decompiler WingDis	21
B'.0.9	O decompiler CFR	21
B'.0.10	O decompiler Procyon	22
B'.0.11	O decompiler FernFlower	22
Γ'	ΤΟ ΣΥΝΤΑΚΤΙΚΟ ΔΕΝΤΡΟ	23
Γ'.1	ΤΟ ΣΥΝΤΑΚΤΙΚΟ ΔΕΝΤΡΟ	23
Γ'.2	ΠΑΡΑΔΕΙΓΜΑΤΑ ΣΥΝΤΑΚΤΙΚΩΝ ΔΕΝΤΡΩΝ	23
Δ'	ΤΕΚΜΗΡΙΩΣΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ ΓΙΑ ΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΤΗ	61
Δ'.1	ΟΙ ΚΛΑΣΕΙΣ ΠΟΥ ΠΕΡΙΕΧΕΙ Ο ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ	61
Δ'.2	ΟΙ ΚΛΑΣΕΙΣ ΣΤΗΝ ΛΕΠΤΟΜΕΡΕΙΑ ΤΟΥΣ	62
Δ'.2.1	Η κλάση Main	62
Δ'.2.2	Η κλάση SourceCodeFileFilter	62
Δ'.2.3	Η κλάση actionToPerform	63
Δ'.2.4	Η κλάση BytecodeConversion	63
Δ'.2.5	Η κλάση ReservedWordsPrinter	64
Δ'.2.6	Η κλάση SymbolTable	64
Δ'.2.7	Η κλάση JavaConversion	65
Δ'.2.8	Η κλάση AbstractSyntaxTreeNode	70
Δ'.2.9	Η κλάση AbstractSyntaxTree	70
Ε'	Ο ΑΛΓΟΡΙΘΜΟΣ	72
Ε'.1	Η ΒΙΒΛΙΟΘΗΚΗ JAVASSIST	72
Ε'.2	ΤΟ ΓΡΑΦΙΚΟ ΠΕΡΙΒΑΛΛΟΝ	74
Ε'.3	Η ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ BYTECODE VIEWER	75
Ε'.4	Ο DECOMPILER	78
Ε'.5	ΤΟ ΣΥΝΤΑΚΤΙΚΟ ΔΕΝΤΡΟ	102

Περιεχόμενα	v
Γ΄ ΠΑΡΑΔΕΙΓΜΑΤΑ	105
Ζ΄ ΕΓΧΕΙΡΙΔΙΟ ΧΡΗΣΗΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ	164
Ζ'.1 ΠΕΡΙΓΡΑΦΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ	164
Ζ'.2 ΕΓΚΑΤΑΣΤΑΣΗ ΚΑΙ ΧΡΗΣΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ	164
Ζ'.3 ΔΕΔΟΜΕΝΑ ΕΞΟΔΟΥ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ	168
Παράρτημα Α΄: Πηγαίος Κώδικας	169
Βιβλιογραφία	238

Κεφάλαιο Α΄

ΕΙΣΑΓΩΓΗ

Στο κεφάλαιο αυτό θα γίνει αναφορά στις βασικές γνώσεις που είναι απαραίτητες ώστε κάποιος να είναι σε θέση να κατανοήσει το περιεχόμενο αυτής της πτυχιακής εργασίας. Αυτές οι γνώσεις αφορούν τις δομές δεδομένων που χρησιμοποιούνται και τον τρόπο αποθήκευσης των δεδομένων κατά τη διάρκεια εκτέλεσης ενός προγράμματος Java, καθώς και τη δομή ενός .class αρχείου.

Α΄.1 Η ιστορία της Java

Στις αρχές του 1991, η Sun αναζητούσε το κατάλληλο εργαλείο για να αποτελέσει την πλατφόρμα ανάπτυξης λογισμικού σε μικρο-συσκευές (έξυπνες οικιακές συσκευές έως πολύπλοκα συστήματα παραγωγής γραφικών). Τα εργαλεία της εποχής ήταν γλώσσες όπως η C++ και η C. Μετά από διάφορους πειραματισμούς προέκυψε το συμπέρασμα ότι οι υπάρχουσες γλώσσες δεν μπορούσαν να καλύψουν τις ανάγκες τους. Ο 'πατέρας' της Java, James Gosling, που εργαζόταν εκείνη την εποχή για την Sun, έκανε ήδη πειραματισμούς πάνω στη C++ και είχε παρουσιάσει κατά καιρούς κάποιες πειραματικές γλώσσες (C++ ++) ως πρότυπα για το νέο εργαλείο που αναζητούσαν στην Sun. Τελικά μετά από λίγο καιρό κατέληξαν με μια πρόταση για το επιτελείο της εταιρίας, η οποία ήταν η γλώσσα Oak. Το όνομά της το πήρε από το ομώνυμο δένδρο (βελανιδιά) το οποίο ο Gosling είχε έξω από το γραφείο του και έβλεπε κάθε μέρα.

Η Oak ήταν μία γλώσσα που διατηρούσε μεγάλη συγγένεια με την C++. Παρόλα αυτά είχε πολύ πιο έντονο αντικειμενοστρεφή (object oriented) χαρακτήρα σε σχέση με την C++ και

χαρακτηριζόταν για την απλότητα της. Σύντομα οι υπεύθυνοι ανάπτυξης της νέας γλώσσας ανακάλυψαν ότι το όνομα Oak ήταν ήδη κατοχυρωμένο οπότε κατά την διάρκεια μιας εκ των πολλών συναντήσεων σε κάποιο τοπικό καφέ αποφάσισαν να μετονομάσουν το νέο τους δημιούργημα σε Java που εκτός των άλλων ήταν το όνομα αγαπητού καφέ για τους δημιουργούς της (Java στην αγγλική γλώσσα είναι το φυτό που βγάζει τον καφέ). Η επίσημη εμφάνιση της Java αλλά και του HotJava (πλοηγός με υποστήριξη Java) στη βιομηχανία της πληροφορικής έγινε το Μάρτιο του 1995 όταν η Sun την ανακοίνωσε στο συνέδριο Sun World 1995. Ο πρώτος μεταγλωττιστής (compiler) της ήταν γραμμένος στη γλώσσα C από τον James Gosling. Το 1994, ο A. Van Hoff ξαναγράφει τον μεταγλωττιστή της γλώσσας σε Java, ενώ το Δεκέμβριο του 1995 πρώτες οι IBM, Borland, Mitsubishi Electronics, Sybase και Symantec ανακοινώνουν σχέδια να χρησιμοποιήσουν τη Java για την δημιουργία λογισμικού. Από εκεί και πέρα η Java ακολουθεί μία ανοδική πορεία και είναι πλέον μία από τις πιο δημοφιλείς γλώσσες στον χώρο της πληροφορικής. Στις 13 Νοεμβρίου του 2006 η Java έγινε πλέον μια γλώσσα ανοιχτού κώδικα (GPL) όσον αφορά το μεταγλωττιστή (javac) και το πακέτο ανάπτυξης (JDK, Java Development Kit).

Στις 27 Απριλίου 2010 η εταιρία λογισμικού Oracle Corporation ανακοίνωσε ότι μετά από πολύμηνες συζητήσεις ήρθε σε συμφωνία για την εξαγορά της Sun Microsystems και των τεχνολογιών (πνευματικά δικαιώματα/ πατέντες) που η δεύτερη είχε στην κατοχή της ή δημιουργήσει. Η συγκεκριμένη συμφωνία θεωρείται σημαντική για το μέλλον της Java και του γενικότερου οικοσυστήματος τεχνολογιών γύρω από αυτή μιας και ο έμμεσος έλεγχος της τεχνολογίας και η εξέλιξη της περνάει σε άλλα χέρια.

Ένα από τα βασικά πλεονεκτήματα της Java έναντι των περισσότερων άλλων γλωσσών είναι η ανεξαρτησία του λειτουργικού συστήματος και πλατφόρμας. Τα προγράμματα που είναι γραμμένα σε Java τρέχουν ακριβώς το ίδιο σε Windows, Linux, Unix και Macintosh (σύντομα θα τρέχουν και σε Playstation καθώς και σε άλλες κονσόλες παιχνιδιών) χωρίς να χρειαστεί να ξαναγίνει μεταγλώττιση (compiling) ή να αλλάξει ο πηγαίος κώδικας για κάθε διαφορετικό λειτουργικό σύστημα. Για να επιτευχθεί όμως αυτό χρειαζόταν κάποιος τρόπος έτσι ώστε τα προγράμματα γραμμένα σε Java να μπορούν να είναι «κατανοητά» από κάθε υπολογιστή ανεξάρτητα του είδους επεξεργαστή (Intel x86, IBM, Sun SPARC, Motorola) αλλά και λειτουργικού συστήματος (Windows, Unix, Linux, BSD, MacOS). Ο λόγος είναι ότι κάθε κεντρική μονάδα επεξεργασίας κατανοεί διαφορετικό κώδικα μηχανής. Ο συμβολικός κώδικας (assembly) που μεταφράζεται και εκτελείται σε Windows είναι διαφορετικός από αυτόν που μεταφράζεται και εκτελείται σε έναν υπολογιστή Macintosh. Η λύση δόθηκε με την ανάπτυξη

της Εικονικής Μηχανής (Virtual Machine ή VM ή EM στα ελληνικά).

Αφού γραφεί κάποιο πρόγραμμα σε Java, στη συνέχεια μεταγλωττίζεται μέσω του μεταγλωττιστή `javac`, ο οποίος παράγει έναν αριθμό από αρχεία `.class` (κώδικας `byte` ή `bytecode`). Ο κώδικας `byte` είναι η μορφή που παίρνει ο πηγαίος κώδικας της Java όταν μεταγλωττίζεται. Όταν πρόκειται να εκτελεστεί η εφαρμογή σε ένα μηχάνημα, το Java Virtual Machine που πρέπει να είναι εγκατεστημένο σε αυτό θα αναλάβει να διαβάσει τα αρχεία `.class`. Στη συνέχεια τα μεταφράζει σε γλώσσα μηχανής που να υποστηρίζεται από το λειτουργικό σύστημα και τον επεξεργαστή, έτσι ώστε να εκτελεστεί (να σημειωθεί εδώ ότι αυτό συμβαίνει με την παραδοσιακή Εικονική Μηχανή (Virtual Machine)). Πιο σύγχρονες εφαρμογές της εικονικής Μηχανής μπορούν και μεταγλωττίζουν εκ των προτέρων τμήματα `bytecode` απευθείας σε κώδικα μηχανής (εγγενή κώδικα ή `native code`) με αποτέλεσμα να βελτιώνεται η ταχύτητα). Χωρίς αυτό δε θα ήταν δυνατή η εκτέλεση λογισμικού γραμμένου σε Java. Πρέπει να σημειωθεί ότι η JVM είναι λογισμικό που εξαρτάται από την πλατφόρμα, δηλαδή για κάθε είδος λειτουργικού συστήματος και αρχιτεκτονικής επεξεργαστή υπάρχει διαφορετική έκδοση του. Έτσι υπάρχουν διαφορετικές JVM για Windows, Linux, Unix, Macintosh, κινητά τηλέφωνα, παιχνιδιομηχανές κλπ.

Οτιδήποτε θέλει να κάνει ο προγραμματιστής (ή ο χρήστης) γίνεται μέσω της εικονικής μηχανής. Αυτό βοηθάει στο να υπάρχει μεγαλύτερη ασφάλεια στο σύστημα γιατί η εικονική μηχανή είναι υπεύθυνη για την επικοινωνία χρήστη - υπολογιστή. Ο προγραμματιστής δεν μπορεί να γράψει κώδικα ο οποίος θα έχει καταστροφικά αποτελέσματα για τον υπολογιστή γιατί η εικονική μηχανή θα τον ανιχνεύσει και δε θα επιτρέψει να εκτελεστεί. Από την άλλη μεριά ούτε ο χρήστης μπορεί να κατεβάσει «κακό» κώδικα από το δίκτυο και να τον εκτελέσει. Αυτό είναι ιδιαίτερα χρήσιμο για μεγάλα κατανεμημένα συστήματα όπου πολλοί χρήστες χρησιμοποιούν το ίδιο πρόγραμμα συγχρόνως.

Ακόμα μία ιδέα που βρίσκεται πίσω από τη Java είναι η ύπαρξη του συλλέκτη απορριμμάτων (Garbage Collector). Συλλογή απορριμμάτων είναι μία κοινή ονομασία που χρησιμοποιείται στον τομέα της πληροφορικής για να δηλώσει την ελευθέρωση τμημάτων μνήμης από δεδομένα που δε χρειάζονται και δε χρησιμοποιούνται άλλο. Αυτή η απελευθέρωση μνήμης στη Java είναι αυτόματη και γίνεται μέσω του συλλέκτη απορριμμάτων. Υπεύθυνη για αυτό είναι και πάλι η εικονική μηχανή η οποία μόλις «καταλάβει» ότι ο σωρός (heap) της μνήμης (στη Java η συντριπτική πλειοψηφία των αντικειμένων αποθηκεύονται στο σωρό σε αντίθεση με τη C++ όπου αποθηκεύονται κυρίως στη στοίβα) κοντεύει να γεμίσει ενεργοποιεί το συλλέκτη απορριμμάτων. Έτσι ο προγραμματιστής δε χρειάζεται να ανησυχεί για το πότε και αν θα

ελευθερώσει ένα συγκεκριμένο τμήμα της μνήμης, ούτε και για σφάλματα δεικτών. Αυτό είναι ιδιαίτερα σημαντικό γιατί είναι κοινά τα σφάλματα προγραμμάτων που οφείλονται σε λανθασμένο χειρισμό της μνήμης.

Παρόλο που η εικονική μηχανή προσφέρει όλα αυτά (και όχι μόνο) τα πλεονεκτήματα, η Java αρχικά ήταν πιο αργή σε σχέση με άλλες προγραμματιστικές γλώσσες υψηλού επιπέδου (high-level) όπως η C και η C++. Εμπειρικές μετρήσεις στο παρελθόν είχαν δείξει ότι η C++ μπορούσε να είναι αρκετές φορές γρηγορότερη από την Java. Ωστόσο γίνονται προσπάθειες από τη Sun για τη βελτιστοποίηση της εικονικής μηχανής, ενώ υπάρχουν και άλλες υλοποιήσεις της εικονικής μηχανής από διάφορες εταιρίες (όπως της IBM), οι οποίες μπορεί σε κάποια σημεία να προσφέρουν καλύτερα και σε κάποια άλλα χειρότερα αποτελέσματα. Επιπλέον με την καθιέρωση των μεταγλωττιστών JIT (Just In Time), οι οποίοι μετατρέπουν τον κώδικα byte απευθείας σε γλώσσα μηχανής, η διαφορά ταχύτητας από τη C++ έχει μικρύνει κατά πολύ.

Οι τελευταίες εκδόσεις του javac με τη χρήση της τεχνολογίας Hot Spot έχουν καταφέρει αξιόλογες επιδόσεις που πλησιάζουν ή και ξεπερνούν σε μερικές περιπτώσεις τον εγγενή κώδικα. [1]

Α'.2 Η δομή της εικονικής μηχανής της Java

Α'.2.1 Τύποι δεδομένων

Η εικονική μηχανή της Java δρα όπως και η γλώσσα προγραμματισμού Java σε δυο τύπους δεδομένων: τους πρωτογενείς και τους τύπους αναφοράς. Αυτό με άλλα λόγια σημαίνει ότι δύο είδη τιμών μπορούν να αποθηκευτούν σε μια μεταβλητή, να περαστούν σαν ορίσματα, να επιστραφούν απο μεθόδους και να υποστούν επεξεργασία, οι πρωτογενείς τιμές και οι αναφορές.

Ο έλεγχος που αφορά τους τύπους δεδομένων γίνεται από κάποιον μεταγλωττιστή και όχι από την ίδια την εικονική μηχανή. Η εικονική μηχανή της Java παρέχει άμεση υποστήριξη των αντικειμένων. Ένα αντικείμενο είναι είτε ένα στιγμιότυπο κλάσης το οποίο καταχωρεί δυναμικά τη μνήμη που του αναλογεί, είτε είναι ένας πίνακας. Οι τιμές τύπου αναφοράς είναι ανάλογες με τους δείκτες σε αντικείμενα. Ένα αντικείμενο μπορεί να έχει παραπάνω από μια αναφορά. Οποιαδήποτε επεξεργασία πάνω στα αντικείμενα γίνεται μέσω τιμών τύπου αναφοράς.

Οι πρωτογενείς τύποι δεδομένων που υποστηρίζονται από την εικονική μηχανή της Java είναι οι αριθμητικοί τύποι (numeric types), ο λογικός τύπος (boolean type) και ο τύπος επιστροφής διεύθυνσης (returnAddress type). Οι αριθμητικοί τύποι αποτελούνται από τους ακέραιους τύπους (integral types) και από τους τύπους κινητής υποδιαστολής (floating-point types).

Οι ακέραιοι τύποι είναι οι ακόλουθοι:

- byte που κυμαίνεται από -128 έως 127
- short που κυμαίνεται από -32768 έως 32767
- int που κυμαίνεται από -2147483648 έως 2147483647
- long που κυμαίνεται από -9223372036854775808 έως 9223372036854775807
- char που κυμαίνεται από 0 έως 65535

Οι τύποι κινητής υποδιαστολής είναι:

- float
- double

Ο τύπος επιστροφής διεύθυνσης χρησιμοποιείται από της εντολές jsr, ret, και jsr_w της εικονικής μηχανής. Οι τιμές του τύπου αυτού είναι δείκτες στους opcodes των εντολών της εικονικής μηχανής της Java. Αντίθετα με τους αριθμητικούς πρωτογενείς τύπους, ο τύπος επιστροφής διεύθυνσης δεν αντιστοιχεί σε κανέναν τύπο της γλώσσας Java και συνεπώς δεν μπορεί να τροποποιηθεί από το ίδιο το πρόγραμμα που εκτελείται.

Παρότι η εικονική μηχανή της Java προσφέρει έναν λογικό τύπο, τον boolean, παρέχει πολύ λίγη υποστήριξη για αυτόν. Δεν υπάρχουν εντολές της εικονικής μηχανής οι οποίες να εκτελούν λειτουργίες μόνο σε boolean τιμές. Αντιθέτως, κατά τη μεταγλώττιση, οι τιμές boolean μετατρέπονται σε ακέραιες (int). Η εικονική μηχανή της Java όμως υποστηρίζει άμεσα τους πίνακες boolean. Η εντολή newarray υποστηρίζει τη δημιουργία πινάκων boolean ενώ οι εντολές baload και bastore χρησιμοποιούνται για την πρόσβαση και την αποθήκευση boolean τιμών αντίστοιχα. Η εικονική μηχανή κωδικοποιεί έναν πίνακα boolean, χρησιμοποιώντας το 1 για να αναπαραστήσει την τιμή 'αληθής' και το 0 για να αναπαραστήσει την τιμή 'ψευδής'. Ο μεταγλωττιστής θα πρέπει να χρησιμοποιήσει την ίδια κωδικοποίηση όταν θα κάνει την μετατροπή σε int.

Όσον αφορά τους τύπους αναφοράς, υπάρχουν οι τύποι αναφοράς κλάσεων (class types), οι τύποι αναφοράς πινάκων (array types) και οι τύποι αναφοράς διεπαφών (interface types). Οι τιμές τους είναι αναφορές σε δυναμικά δημιουργημένα στιγμιότυπα κλάσεων, σε πίνακες ή σε δυναμικά δημιουργημένα στιγμιότυπα κλάσεων ή πίνακες που υλοποιούν κάποια διεπαφή. Μια αναφορά μπορεί ακόμα και να έχει την ειδική τιμή null η οποία σημαίνει ότι ο τύπος αυτός δεν αναφέρεται ακόμα σε κανένα αντικείμενο.

Α΄.2.2 Περιοχές δεδομένων κατά τη διάρκεια εκτέλεσης των προγραμμάτων

Η εικονική της Java χρησιμοποιεί διάφορες περιοχές δεδομένων κατά τη διάρκεια εκτέλεσης των προγραμμάτων. Κάποιες από αυτές τις περιοχές δεδομένων δημιουργούνται κατά την εκκίνηση της εικονικής μηχανής και καταστρέφονται μόνο κατά τον τερματισμό της. Άλλες περιοχές δεδομένων αντιστοιχούν σε νήματα. Αυτές δημιουργούνται όταν δημιουργείται το νήμα και έχουν διάρκεια ζωής ίση με αυτή του νήματος.

Α΄.2.2.1 Ο καταχωρητής pc (program counter)

Η εικονική μηχανή της Java μπορεί να υποστηρίξει την εκτέλεση πολλών νημάτων ταυτόχρονα. Κάθε νήμα έχει τον δικό του καταχωρητή pc. Κάθε χρονική στιγμή, κάθε νήμα εκτελεί τον κώδικα μιας μοναδικής μεθόδου, της τρέχουσας μεθόδου αυτού του νήματος. Ο καταχωρητής pc περιέχει τη διεύθυνση της εντολής που εκτελείται από την εικονική μηχανή.

Α΄.2.2.2 Οι στοίβες της εικονικής μηχανής

Κάθε νήμα της εικονικής μηχανής έχει μια στοίβα που είναι private και δημιουργείται την ίδια στιγμή που δημιουργείται και το νήμα. Μια στοίβα της εικονικής μηχανής αποθηκεύει πλαίσια (frames). Μια τέτοια στοίβα είναι ανάλογη με τη στοίβα των συμβατικών γλωσσών όπως η C, δηλαδή αποθηκεύει τοπικές μεταβλητές και μερικά αποτελέσματα και παίζει τον ρόλο στην κλήση συναρτήσεων και στην επιστροφή από αυτές. Τα πλαίσια όμως μπορούν να αποθηκευτούν ακόμα και σε ένα σωρό (heap) επειδή η στοίβα είναι σε θέση να δεχτεί άμεσο χειρισμό εκτός από ωθήσεις και απωθήσεις πλαισίων. Η μνήμη μιας τέτοιας στοίβας δεν χρειάζεται να είναι συνεχής. Οι στοίβες αυτές μπορούν να είναι είτε σταθερού μεγέθους είτε να επεκτείνονται δυναμικά ανάλογα με τις ανάγκες των υπολογισμών. Αν είναι σταθερού

μεγέθους, τότε το μέγεθος μπορεί να επιλεγεί ανεξάρτητα κατά τη δημιουργία τους.

Οι ακόλουθες εξαιρέσεις σχετίζονται με τις στοίβες της εικονικής μηχανής:

- 1) Αν ένας υπολογισμός ενός νήματος απαιτεί μια στοίβα μεγαλύτερο μεγέθους από το επιτρεπόμενο, τότε η εικονική μηχανή εμφανίζει ένα `StackOverflowError`.
- 2) Αν οι στοίβες της εικονικής μηχανής μπορούν να επεκταθούν δυναμικά, επιχειρηθεί επέκταση αλλά δεν υπάρχει επαρκής μνήμη που να καλύπτει την επέκταση, ή υπάρχει ανεπαρκής μνήμη για τη δημιουργία της αρχικής στοίβας ενός νέου νήματος, τότε η εικονική μηχανή εμφανίζει ένα `OutOfMemoryError`.

Α'.2.2.3 Ο σωρός (Heap)

Ο σωρός της εικονικής μηχανής της Java χρησιμοποιείται από όλα τα νήματα της. Ο σωρός είναι η περιοχή δεδομένων από την οποία αντλούν μνήμη όλα τα στιγμιότυπα των κλάσεων και των πινάκων κατά τη διάρκεια της εκτέλεσης. Ο σωρός δημιουργείται κατά την εκκίνηση της εικονικής μηχανής. Η αποθήκευση αντικειμένων στο σωρό γίνεται μέσω ενός αυτόματου συστήματος διαχείρισης της μνήμης που είναι γνωστό ως `garbage collector`. Ο σωρός μπορεί να έχει σταθερό μέγεθος ή μπορεί να είναι επεκτάσιμος ανάλογα με τις απαιτήσεις των υπολογισμών ή ακόμα μπορεί και να συμπιεστεί αν το μέγεθος του δεν είναι χρήσιμο. Η μνήμη του σωρού δεν χρειάζεται να είναι συνεχής. Αν ένας υπολογισμός απαιτεί ο σωρός να έχει μεγαλύτερη μνήμη από αυτήν που μπορεί να διαθέσει ο αυτόματος διαχειριστής μνήμης, τότε η εικονική μηχανή εμφανίζει ένα `OutOfMemoryError`.

Α'.2.2.4 Η περιοχή των μεθόδων (Method Area)

Η εικονική μηχανή έχει μια περιοχή μεθόδων η οποία χρησιμοποιείται από όλα τα νήματα. Η περιοχή μεθόδων είναι ανάλογη με την περιοχή αποθήκευσης του μεταγλωττισμένου κώδικα μιας συμβατικής γλώσσας. Αποθηκεύει δομές ανά κλάση όπως το `runtime constant pool`, δεδομένα πεδίων και μεθόδων και τον κώδικα των μεθόδων και των `constructors` αλλά και των ειδικών μεθόδων που χρησιμοποιούνται κατά την αρχικοποίηση κλάσεων, διεπαφών και στιγμιότυπων. Η περιοχή μεθόδων δημιουργείται κατά την εκκίνηση της εικονικής μηχανής και μπορεί να είναι είτε σταθερού μεγέθους είτε επεκτάσιμη ανάλογα με τις ανάγκες των υπολογισμών και μπορεί ακόμα και να συμπιεστεί αν μέρος της μνήμης της δεν χρησιμοποιείται. Η μνήμη που αντιστοιχεί στην περιοχή μεθόδων δεν χρειάζεται να είναι συνεχόμενη. Αν η

μνήμη της περιοχής μεθόδων δεν είναι αρκετή για να ικανοποιήσει κάποιο αίτημα διάθεσης μνήμης, τότε η εικονική μνήμη εμφανίζει ένα `OutOfMemoryError`.

Α'.2.2.5 Runtime Constant Pool

Το runtime constant pool είναι μια αναπαράσταση του πίνακα `constant_pool` ενός αρχείου class ανά κλάση ή ανα διεπαφή. Περιέχει διάφορα είδη σταθερών, από αριθμητικές ως και αναφορές σε πεδία και μεθόδους οι οποίες πρέπει να αναλυθούν κατά τη διάρκεια της εκτέλεσης. Το runtime constant pool έχει λειτουργία παρόμοια με αυτή του πίνακα συμβόλων (`symbol table`) που χρησιμοποιείται από τις συμβατικές γλώσσες προγραμματισμού, παρά το γεγονός ότι περιέχει ένα μεγαλύτερο εύρος δεδομένων από ότι ο πίνακας συμβόλων. Κάθε runtime constant pool αντλεί μνήμη από την περιοχή μεθόδων της εικονικής μηχανής. Το runtime constant pool για μια κλάση ή μια διεπαφή δημιουργείται από την εικονική μηχανή κατά τη δημιουργία της κλάσης ή της διεπαφής. Αν κατά τη δημιουργία μιας κλάσης ή μιας διεπαφής, η δημιουργία του runtime constant pool απαιτεί περισσότερη μνήμη από αυτή που είναι διαθέσιμη στην περιοχή μεθόδων της εικονικής μηχανής, τότε η εικονική μηχανή εμφανίζει ένα `OutOfMemoryError`.

Α'.2.3 Τα πλαίσια (frames)

Ένα πλαίσιο χρησιμοποιείται για την αποθήκευση δεδομένων και μερικών αποτελεσμάτων, αλλά πραγματοποιεί και δυναμική σύνδεση, επιστροφή τιμών μεθόδων και χειρισμό εξαιρέσεων. Κάθε φορά που καλείται μια μέθοδος δημιουργείται ένα νέο πλαίσιο. Το πλαίσιο καταστρέφεται όταν η κλήση της μεθόδου του ολοκληρώνεται είτε κανονικά είτε απότομα (δηλαδή παρουσιάζεται εξαίρεση). Τα πλαίσια αντλούν την μνήμη τους από τη στοίβα του νήματος που δημιούργησε το πλαίσιο. Κάθε πλαίσιο έχει τον δικό του πίνακα τοπικών μεταβλητών, τη δική του στοίβα τελεστών και τη δική του αναφορά στο runtime constant pool της κλάσης στην οποία αντιστοιχεί η τρέχουσα μέθοδος. Το μέγεθος του πίνακα τοπικών μεταβλητών και της στοίβας τελεστών καθορίζονται τη στιγμή της μεταγλώττισης και παρέχονται μαζί με τον κώδικα της μεθόδου που σχετίζεται με το πλαίσιο. Το μέγεθος όμως των δομών δεδομένων του πλαισίου εξαρτάται από την υλοποίηση της εικονικής μηχανής. Επισημαίνεται επίσης ότι για κάθε νήμα, μόνο το πλαίσιο της μεθόδου που εκτελείται είναι ενεργό κάθε χρονική στιγμή. Το πλαίσιο αυτό αναφέρεται ως τρέχον πλαίσιο και η μέθοδος του είναι γνωστή ως τρέχουσα μέθοδος. Η κλάση στην οποία έχει οριστεί η τρέχουσα μέθοδος ορίζεται ως η τρέχουσα

κλάση. Οι λειτουργίες σε τοπικές μεταβλητές και στη στοίβα τελεστών αναφέρονται μαζί με το τρέχον πλαίσιο.

Ένα πλαίσιο παύει να είναι το τρέχον αν η μέθοδος του καλέσει κάποια άλλη μέθοδο ή αν η μέθοδος του ολοκληρωθεί. Όταν καλείται μια μέθοδος, δημιουργείται ένα νέο πλαίσιο το οποίο γίνεται τρέχον όταν ο έλεγχος μεταφερθεί στη μέθοδο αυτή. Κατά την επιστροφή από μια μέθοδο, το τρέχον πλαίσιο περνά το αποτέλεσμα από την κλήση της μεθόδου (αν υπάρχει) στο προηγούμενο πλαίσιο. Το τρέχον πλαίσιο τότε αποβάλλεται, και το προηγούμενο πλαίσιο γίνεται ξανά τρέχον.

Α'.2.3.1 Τοπικές μεταβλητές (Local Variables)

Κάθε πλαίσιο περιέχει έναν πίνακα μεταβλητών γνωστών ως τοπικές μεταβλητές. Το μέγεθος του πίνακα τοπικών μεταβλητών ενός πλαισίου καθορίζεται κατά τη μεταγλώττιση και παρέχεται στη δυαδική αναπαράσταση μιας κλάσης ή μιας διεπαφής μαζί με τον κώδικα της μεθόδου με την οποία σχετίζεται το πλαίσιο. Μια τοπική μεταβλητή μπορεί να περιέχει μια τιμή τύπου boolean, byte, char, short, int, float, reference, ή returnAddress. Ένα ζευγάρι τοπικών μεταβλητών μπορεί να περιέχει μια τιμή τύπου long ή double. Οι τοπικές μεταβλητές διευθυνσιοδοτούνται με δείκτες (indexes). Ο δείκτης της πρώτης τοπικής μεταβλητής είναι το 0. Η τιμή του δείκτη πρέπει να κυμαίνεται από το 0 ως το μέγεθος του πίνακα τοπικών μεταβλητών μείον 1. Μια τιμή τύπου long ή double καταλαμβάνει δύο συνεχόμενες τοπικές μεταβλητές. Αυτό σημαίνει ότι αν έχουμε για παράδειγμα μια τιμή τύπου double η οποία είναι αποθηκευμένη στον πίνακα τοπικών μεταβλητών στη θέση n, στην πραγματικότητα καταλαμβάνει τις τοπικές μεταβλητές με δείκτες n και n+1. Άρα, η τοπική μεταβλητή με δείκτη n+1 δεν μπορεί να φορτωθεί. Μπορεί όμως να δεχτεί την πράξη της αποθήκευσης, κάτι τέτοιο όμως θα ακύρωνε το περιεχόμενο της τοπικής μεταβλητής n.

Η εικονική μηχανή της Java χρησιμοποιεί τοπικές μεταβλητές για να περάσει τις παραμέτρους κατά την κλήση των μεθόδων. Κατά την κλήση μιας μεθόδου, οποιαδήποτε παράμετρος περνά σε συνεχόμενες τοπικές μεταβλητές ξεκινώντας από την τοπική μεταβλητή 0. Όταν καλείται μια μέθοδος ενός στιγμιοτύπου μιας κλάσης, η τοπική μεταβλητή 0 χρησιμοποιείται πάντα για να περάσει ως παράμετρο μια αναφορά στο αντικείμενο το οποίο καλεί τη μέθοδο (δηλαδή το γνωστό this). Οι υπόλοιποι παράμετροι περνούν μέσω συνεχόμενων τοπικών μεταβλητών ξεκινώντας από την τοπική μεταβλητή 1.

Α'.2.3.2 Στοιίβες Δεδομένων (Operand Stacks)

Κάθε πλαίσιο περιέχει μια LIFO στοίβα τελεστών. Το μέγεθος της στοίβας αυτής καθορίζεται τη στιγμή της μεταγλώττισης και παρέχεται μαζί με τον κώδικα της μεθόδου που σχετίζεται με το πλαίσιο. Η στοίβα είναι άδεια όταν δημιουργείται το πλαίσιο που την περιέχει. Η εικονική μηχανή παρέχει εντολές που αφορούν τη φόρτωση σταθερών, τιμών τοπικών μεταβλητών ή πεδίων στην στοίβα τελεστών. Άλλες εντολές της εικονικής μηχανής παίρνουν τελεστές από τη στοίβα τελεστών, τους επεξεργάζονται και μετά ωθούν το αποτέλεσμα στη στοίβα. Η στοίβα τελεστών χρησιμοποιείται επίσης για την προετοιμασία των παραμέτρων που πρέπει να περαστούν σε μεθόδους και για να λάβουν τα αποτελέσματα των μεθόδων. Για παράδειγμα, η εντολή `iadd` προσθέτει δυο ακέραιες τιμές. Η εντολή αυτή απαιτεί οι δυο ακέραιες τιμές που θα προστεθούν να βρίσκονται στην κορυφή της στοίβας των τελεστών. Και οι δυο τιμές απωθούνται από τη στοίβα τελεστών, προστίθενται μεταξύ τους και το άθροισμα τους ωθείται στη στοίβα τελεστών. Κάθε θέση της στοίβας τελεστών μπορεί να αποθηκεύσει τιμές οποιουδήποτε τύπου της εικονικής μηχανής, συμπεριλαμβανομένων και των τύπων `long` και `double`.

Οι τιμές της στοίβας τελεστών επιβάλλεται να δέχονται επεξεργασία με τρόπους κατάλληλους του τύπου τους. Για παράδειγμα, δεν είναι δυνατόν να ωθηθούν δύο ακέραιες τιμές και να συμπεριφερθούμε σε αυτές σαν να ήταν τύπου `long` ή να ωθηθούν δύο τιμές `float` και να τις αθροίσουμε σαν να ήταν ακέραιες με την εντολή `iadd`. Ένας μικρός αριθμός εντολών της εικονικής μηχανής (οι εντολές `dup` και `swap`) δρουν στις τιμές ανεξάρτητα του τύπου τους. Αυτές οι εντολές όμως έχουν καθοριστεί έτσι που δεν μπορούν να τροποποιήσουν τις τιμές. Αυτοί οι περιορισμοί όσον αφορά τον χειρισμό της στοίβας τελεστών επιβάλλονται μέσω της διαδικασίας επαλήθευσης του `class` αρχείου. Τέλος, επισημαίνεται ότι μια τιμή τύπου `long` ή `double` καταλαμβάνει δύο συνεχόμενες θέσεις στη στοίβα τελεστών ενώ μια τιμή οποιουδήποτε άλλου τύπου καταλαμβάνει μία μόνο θέση.

Α'.2.3.3 Ειδικές μέθοδοι αρχικοποίησης

Στο επίπεδο της εικονικής μηχανής κάθε constructor εμφανίζεται σαν μια μέθοδος αρχικοποίησης στιγμιότυπων με το ειδικό όνομα `<init>`. Το όνομα `<init>` δεν είναι έγκυρο, δεν μπορεί να χρησιμοποιηθεί ευθέως σε ένα πρόγραμμα γραμμένο σε Java. Οι μέθοδοι αρχικοποίησης καλούνται μόνο μέσα στην εικονική μηχανή για μη αρχικοποιημένα στιγμιότυπα, μέσω της ειδικής εντολής `invokespecial`. Μια κλάση ή μια διεπαφή έχει το πολύ μια μέθοδο

αρχικοποίησης η οποία είναι static και δεν παίρνει ορίσματα. Η μέθοδος αυτή έχει το ειδικό όνομα <clinit>. Το όνομα <clinit> δεν είναι έγκυρο, δεν μπορεί να χρησιμοποιηθεί ευθέως σε ένα πρόγραμμα γραμμένο σε Java. Οι μέθοδοι αρχικοποίησης μιας κλάσης ή μιας διεπαφής καλούνται έμμεσα από την εικονική μηχανή σαν μέρος της διαδικασίας αρχικοποίησης της κλάσης και δεν καλούνται ποτέ άμεσα μέσω κάποιας εντολής της εικονικής μηχανής. [2]

Α΄.3 Η δομή και η διάταξη ενός class αρχείου

Α΄.3.1 Η δομή του class αρχείου

Κάθε αρχείο class περιέχει τον ορισμό μιας κλάσης ή μιας διεπαφής. Ένα αρχείο class αποτελείται από μια ροή bytes των 8 bit. Όλες οι πολλαπλάσιες ποσότητες bit (16, 32, 64 bit) κατασκευάζονται διαβάζοντας συνεχόμενες οχτάδες bit (2, 4, 8 οχτάδες αντίστοιχα). Τα αντικείμενα δεδομένων τα οποία χρειάζονται πάνω από ένα byte για την αναπαράστασή τους, αποθηκεύονται σε big-endian μορφή.

Η δομή ενός class αρχείου είναι η ακόλουθη:

```
ClassFile {  
    u4 magic;  
    u2 minor_version;  
    u2 major_version;  
    u2 constant_pool_count;  
    cp_info constant_pool[constant_pool_count-1];  
    u2 access_flags;  
    u2 this_class;  
    u2 super_class;  
    u2 interfaces_count;  
    u2 interfaces[interfaces_count];  
    u2 fields_count;  
    field_info fields[fields_count];  
    u2 methods_count;  
    method_info methods[methods_count];  
    u2 attributes_count;  
    attribute_info attributes[attributes_count];  
}
```

}

Τα περιεχόμενα της δομής αυτής είναι τα ακόλουθα:

magic: προσδιορίζει τη μορφή και τη διάταξη (format) του class αρχείου και έχει την τιμή 0xCAFEBADE.

minor_version, major_version: αυτές οι δύο τιμές προσδιορίζουν την έκδοση της μορφής και της διάταξης (format) του class αρχείου. Αν ένα class αρχείο έχει major version M και minor version m, τότε η έκδοση του σημειώνεται ως M.m.

constant_pool_count: η τιμή αυτή ισούται με τον αριθμό των εγγραφών στον πίνακα constant_pool. Ένας δείκτης θέσης του πίνακα constant_pool είναι έγκυρος αν είναι ανάμεσα στο 0 και το constant_poo_count.

constant_pool[]: είναι ένας πίνακας δομών που αναπαριστούν διάφορες αλφαριθμητικές σταθερές, ονόματα κλάσεων και διεπαφών, ονόματα πεδίων και άλλες σταθερές που αναφέρονται στο αρχείο. Ο πίνακας constant_pool ξεκινάει από τη θέση 0 και εκτείνεται ως τη θέση constant_pool.count-1.

access_flags: είναι μια μάσκα τιμών που χρησιμοποιείται για να δείξει τα δικαιώματα που υπάρχουν και αφορούν αυτήν την κλάση ή τη διεπαφή. Οι τιμές είναι οι ακόλουθες:

ACC_PUBLIC: επιτρέπεται η πρόσβαση εκτός πακέτου

ACC_FINAL: δεν επιτρέπονται υποκλάσεις

ACC_SUPER: να υπάρχει ειδική μεταχείριση των μεθόδων των υπερκλάσεων όταν καλούνται από την εντολή invokespecial

ACC_INTERFACE: πρόκειται για διεπαφή και όχι για κλάση

ACC_ABSTRACT: η κλάση είναι abstract άρα δεν μπορεί να δημιουργηθεί στιγμιότυπο της

Μια διεπαφή ξεχωρίζει αν έχει την τιμή ACC_INTERFACE. Αν το class αρχείο δεν έχει αυτήν την τιμή, τότε πρόκειται για κλάση και όχι για διεπαφή. Αν υπάρχει η τιμή ACC_INTERFACE τότε πρέπει να υπάρχει και η τιμή ACC_ABSTRACT αλλά και η ACC_PUBLIC. Αν δεν υπάρχει η τιμή ACC_INTERFACE τότε μπορεί να υπάρχει οποιαδήποτε από τις άλλες τιμές αρκεί να μην υπάρχουν ταυτόχρονα οι τιμές ACC_FINAL και ACC_ABSTRACT.

this_class: η τιμή αυτή πρέπει να είναι ένας έγκυρος δείκτης θέσης στον πίνακα constant_pool. Η εγγραφή σε αυτή τη θέση πρέπει να είναι μια δομή CONSTANT_Class_info η οποία αναπαριστά την κλάση ή τη διεπαφή που ορίζεται στο αρχείο.

super_class: η τιμή αυτή πρέπει ή να είναι 0 ή να είναι ένας έγκυρος δείκτης θέσης στον πίνακα

`constant_pool`. Αν η τιμή δεν είναι 0, τότε η εγγραφή του `constant_pool` σε αυτή τη θέση πρέπει να είναι μια δομή `CONSTANT_Class_info` που να αναπαριστά την άμεση υπερκλάση της κλάσης που ορίζεται από το αρχείο. Ούτε η άμεση υπερκλάση ούτε καμία από τις υπερκλάσεις της δεν πρέπει να είναι `final`. Αν η τιμή είναι 0, τότε η κλάση που ορίζει το αρχείο πρέπει να είναι η κλάση `Object`, η μοναδική κλάση χωρίς άμεση υπερκλάση. Για μια διεπαφή, η τιμή αυτή πρέπει να είναι πάντα ένας έγκυρος δείκτης θέσης στον πίνακα `constant_pool`. Η εγγραφή του πίνακα `constant_pool` σε αυτή τη θέση πρέπει να είναι μια δομή `CONSTANT_Class_info` που να αναπαριστά την κλάση `Object`.

interfaces_count: η τιμή αυτή δίνει το πλήθος των άμεσων υπερδιεπαφών αυτής της κλάσης ή της διεπαφής.

interfaces[]: Κάθε τιμή του πίνακα των διεπαφών πρέπει να είναι ένας έγκυρος δείκτης θέσης στον πίνακα `constant_pool`. Η εγγραφή του πίνακα `constant_pool` στη θέση `interfaces[i]`, όπου το `i` πρέπει να βρίσκεται ανάμεσα στο 0 και το `interfaces_count`, πρέπει να είναι μια δομή `CONSTANT_Class_info` που να αναπαριστά μια διεπαφή που να είναι άμεση υπερδιεπαφή της κλάσης ή της διεπαφής που ορίζεται από το αρχείο.

fields_count: η τιμή αυτή περιέχει το πλήθος των δομών `field_info` που υπάρχουν στον πίνακα `fields`. Οι δομές αναπαριστούν όλα τα πεδία, τις μεταβλητές κλάσης και τις μεταβλητές στιγμιοτύπου που δηλώνονται στην κλάση ή τη διεπαφή.

fields[]: Κάθε τιμή του πίνακα πρέπει να είναι μια δομή `field_info` η οποία να δίνει μια ολοκληρωμένη περιγραφή ενός πεδίου αυτής της κλάσης ή της διεπαφής. Ο πίνακας αυτός περιέχει μόνο αυτά τα πεδία τα οποία έχουν δηλωθεί στην κλάση ή τη διεπαφή. Δεν περιέχει πεδία τα οποία κληρονομούνται από τις υπερκλάσεις ή τις υπερδιεπαφές.

methods_count: Η τιμή αυτή περιέχει το πλήθος των δομών `method_info` structures που υπάρχουν στον πίνακα `methods`.

methods[]: Κάθε τιμή του πίνακα `methods` πρέπει να είναι μια δομή `method_info` η οποία να δίνει μια πλήρη περιγραφή μιας μεθόδου της κλάσης ή της διεπαφής που ορίζει το αρχείο. Αν η μέθοδος δεν είναι `abstract` παρέχονται και οι εντολές της εικονικής μηχανής που υλοποιούν τη μέθοδο. Οι δομές `method_info` αναπαριστούν όλες τις μεθόδους που δηλώνονται στην κλάση ή τη διεπαφή συμπεριλαμβανομένων των μεθόδους των στιγμιοτύπων, των μεθόδους των κλάσεων (`static`) και των μεθόδων αρχικοποίησης. Ο πίνακας μεθόδων δεν περιέχει κληρονομούμενες από υπερκλάσεις ή υπερδιεπαφές μεθόδους.

attributes_count: Η τιμή αυτή περιέχει το πλήθος των χαρακτηριστικών που υπάρχουν

στον πίνακα χαρακτηριστικών της κλάσης.

attributes[]: Κάθε τιμή του πίνακα χαρακτηριστικών πρέπει να είναι μια δομή χαρακτηριστικών (όπως για παράδειγμα τα χαρακτηριστικά `SourceFile` και `Deprecated`).

Όσον αφορά τα ονόματα των κλάσεων και τα διεπαφών, επισημαίνεται ότι αυτά εμφανίζονται πλήρη δηλαδή συμπεριλαμβάνουν και τα πακέτα στα οποία ανήκουν. Για παράδειγμα η κλάση `Thread`, σε ένα class αρχείο εμφανίζεται ως `java.lang.Thread`.

A΄.3.2 Descriptors

Ένας descriptor είναι ένα string το οποίο αναπαριστά τον τύπο ενός πεδίου ή μιας μεθόδου χρησιμοποιώντας κωδικοποίηση UTF-8. Οι descriptors ορίζονται χρησιμοποιώντας μια γραμματική. Αυτή η γραμματική είναι ένα σύνολο παραγωγών που περιγράφουν πως ορισμένες ακολουθίες χαρακτήρων μπορούν να σχηματίσουν συντακτικά ορθούς descriptors διαφόρων τύπων. Τα μη τερματικά σύμβολα είναι γραμμένα με *italics* ενώ τα τερματικά όχι. Για παράδειγμα η παραγωγή:

FieldType:

BaseType

ObjectType

ArrayType

δηλώνει ότι το *FieldType* μπορεί να αναπαριστά είτε ένα *BaseType*, είτε ένα *ObjectType* είτε ένα *ArrayType*.

Η παραγωγή:

MethodDescriptor:

(*ParameterDescriptor**) *ReturnDescriptor*

δηλώνει ότι το *MethodDescriptor* αναπαριστά μια αριστερή παρένθεση η οποία ακολουθείται από 0 ή περισσότερες τιμές *ParameterDescriptor*, όπου τέλος ακολουθούν μια δεξιά παρένθεση και μια τιμή *ReturnDescriptor*.

A΄.3.2.1 Descriptors πεδίων

Ένας descriptor πεδίου αναπαριστά τον τύπο μιας κλάσης, ενός στιγμιότυπου ή μιας τοπικής μεταβλητής. Είναι μια σειρά χαρακτήρων που παράγεται από την ακόλουθη γραμματική:

FieldDescriptor:

FieldType

ComponentType:

FieldType

FieldType:

BaseType

ObjectType

ArrayType

BaseType:

B

C

D

F

I

J

S

Z

ObjectType:

L <classname> ;

ArrayType:

[*ComponentType*

Το <classname> αναπαριστά ένα πλήρες όνομα κλάσης ή διεπαφής. Τα σύμβολα BaseType έχουν την εξής σημασία:

B: byte

C: char

D: double

F: float

I: int

J: long

L<classname>; : αναφορά

S: short

Z: boolean

[: αναφορά (μονοδιάστατος πίνακας)

Για παράδειγμα, ο descriptor μιας μεταβλητής τύπου `int` είναι απλά `I`. Ο descriptor μια μεταβλητής τύπου `Object` είναι `Ljava/lang/Object;`. Ο descriptor μιας μεταβλητής που είναι αναφορά σε έναν πολυδιάστατο πίνακα αριθμών `double d[][]` είναι `[[[D`.

Α΄.3.2.2 Descriptors μεθόδων

Ένας descriptor μιας μεθόδου αναπαριστά τις παραμέτρους που παίρνει η μέθοδος και την τιμή που επιστρέφει:

MethodDescriptor:

(ParameterDescriptor) ReturnDescriptor*

Ένας descriptor παραμέτρου αναπαριστά την παράμετρο που περνά σε μια μέθοδο:

ParameterDescriptor:

FieldType

Ένας descriptor επιστροφής αναπαριστά τον τύπο της τιμής που επιστρέφεται από μια μέθοδο. Είναι μια σειρά χαρακτήρων που παράγονται από την γραμματική:

ReturnDescriptor:

FieldType

V

Ο χαρακτήρας `V` δείχνει ότι η μέθοδος δεν επιστρέφει καμία τιμή (ο τύπος επιστροφής της είναι δηλαδή ο `void`).

Για παράδειγμα, ο descriptor της μεθόδου `Object mymethod(int i, double d, Thread t)` είναι `(IDLjava/lang/Thread;)Ljava/lang/Object;` [2]

Κεφάλαιο Β΄

ΣΚΟΠΟΙ ΧΡΗΣΗΣ & ΠΑΡΕΜΦΕΡΗΣ ΕΡΕΥΝΑ



ε αυτό το κεφάλαιο θα γίνει μια αναφορά στους σκοπούς χρήσης των decompilers καθώς και στα υπάρχοντα λογισμικά που υλοποιούν τη λειτουργία του java decompiler.

Β΄.0.1 Σκοποί χρήσης των decompilers

1. Κατανόηση και εντοπισμός σφαλμάτων κώδικα: οι προγραμματιστές χρησιμοποιούν decompilers για να αναλύσουν και να κατανοήσουν τη λειτουργικότητα του μεταγλωττισμένου κώδικα καθώς και των βιβλιοθηκών που χρησιμοποιούν. Αυτό είναι ιδιαίτερα χρήσιμο όταν ο αρχικός πηγαίος κώδικας δεν είναι διαθέσιμος ή όταν πρόκειται για συστήματα παλαιού τύπου. Όσον αφορά την γλώσσα προγραμματισμού Java, τα ολοκληρωμένα προγραμματιστικά περιβάλλοντα όπως το IntelliJ και το Eclipse έχουν ενσωματωμένους decompilers έτσι ώστε να βοηθούν τους προγραμματιστές στην ανάλυση κλάσεων για τις οποίες δεν είναι διαθέσιμος ο πηγαίος κώδικας.

2. Ανάκτηση χαμένου πηγαίου κώδικα: Σε περιπτώσεις όπου ο αρχικός πηγαίος κώδικας έχει χαθεί ή δεν είναι διαθέσιμος, μπορεί να χρησιμοποιηθεί ένας decompiler για την ανάκτηση του κατά προσέγγιση. Αυτό μπορεί να είναι ιδιαίτερα χρήσιμο στην περίπτωση της συντήρησης ή της ενημέρωσης λογισμικού του οποίου ο πηγαίος κώδικας δεν είναι εύκολα προσβάσιμος.

3. Στον τομέα της ασφάλειας: οι decompilers χρησιμοποιούνται επίσης εκτενώς στον τομέα της κυβερνοασφάλειας και της ανάλυσης κακόβουλου λογισμικού. Οι decompilers δίνουν τη δυνατότητα τους ειδικούς του τομέα της κυβερνοασφάλειας να αποκαλύψουν κακόβουλες ενέργειες, συμπεριλαμβανομένης της κλοπής δεδομένων, του τηλεχειρισμού ή της εκμετάλλευσης τρωτών σημείων του συστήματος. Η διαδικασία του decompiling επιτρέπει στους ειδικούς να εξετάσουν εξονυχιστικά την εσωτερική λειτουργία του κακόβουλου λογισμικού, δίνοντας τους τη δυνατότητα να αναπτύξουν αντίμετρα και στρατηγικές για την αφαίρεση του. Επιπλέον, στον τομέα της κυβερνοασφάλειας, είναι σημαντικό να εντοπίζονται και να επιδιορθώνονται τα τρωτά σημεία του συστήματος προτού οι κακόβουλοι χρήστες καταφέρουν να τα εκμεταλλευτούν. Οι decompilers βοηθούν στον εντοπισμό πιθανών τρωτών σημείων. Αναλύοντας τον πηγαίο κώδικα που δημιουργείται από την διαδικασία του decompiling, οι ειδικοί σε θέματα ασφάλειας μπορούν να εντοπίσουν ανασφαλείς πρακτικές προγραμματισμού, όπως υπερχειλίσσεις buffer ή ελαττώματα επικύρωσης εισόδου, και να προτείνουν διορθώσεις για την βελτίωση της ασφάλειας του λογισμικού. Οι decompilers είναι επίσης ένα πολύτιμο εργαλείο για τη συλλογή πληροφοριών που αφορούν τις ίδιες τις απειλές. Οι ερευνητές της κυβερνοασφάλειας μπορούν να απομεταγλωττίσουν δείγματα κακόβουλου λογισμικού για να ανακαλύψουν μοτίβα κώδικα, τεχνικές και τακτικές που χρησιμοποιούνται από επιτιθέμενους. Αυτές οι πληροφορίες μπορούν στη συνέχεια να κοινοποιηθούν στην κοινότητα της κυβερνοασφάλειας με σκοπό τη βελτίωση της άμυνας έναντι παρόμοιων απειλών. Τέλος, οι decompilers μπορούν να βοηθήσουν στην ταχεία απόκριση σε περιστατικά παραβίασης της ασφάλειας στον κυβερνοχώρο, αναλύοντας γρήγορα ύποπτα αρχεία ή εκτελέσιμα. Οι ομάδες ασφαλείας μπορούν να χρησιμοποιήσουν τον απομεταγλωττισμένο κώδικα για να κατανοήσουν τη φύση της επίθεσης και να αναπτύξουν αντίμετρα για να μετριάσουν τη ζημιά και να αποτρέψουν περαιτέρω εισβολές. [16]

4. Εκμετάλλευση λογισμικού: Οι επιτιθέμενοι (hackers) μπορεί να χρησιμοποιήσουν decompilers για να εξετάσουν τον πηγαίο κώδικα ενός προγράμματος, προκειμένου να εντοπίσουν ευπάθειες και να αναπτύξουν επιθέσεις.

5. Αντιγραφή κώδικα: Σε ορισμένες περιπτώσεις, οι decompilers μπορούν να χρησιμοποιηθούν για την αντιγραφή του πηγαίου κώδικα ενός προγράμματος, πράγμα που μπορεί να έχει νομικές επιπτώσεις.

Παρακάτω παρατίθενται ορισμένα υπάρχοντα λογισμικά τα οποία υλοποιούν τη λειτουργία του java decompiler.

B'.0.2 Ο decompiler DAVA

Το πιο γνωστό και πιο λειτουργικό εργαλείο το οποίο έχει αναπτυχθεί είναι ο DAVA. Ο DAVA έχει αναπτυχθεί στο πανεπιστήμιο McGill του Καναδά από τον Jerome Miecznikowski το 2003. Ο DAVA έχει αναπτυχθεί χρησιμοποιώντας το Soot Framework το οποίο είναι και αυτό έργο του ίδιου πανεπιστημίου με σκοπό την ανάλυση και την βελτιστοποίηση των bytecodes πριν τρέξουν στη Java Virtual Machine. Ο DAVA φυσικά είναι ένας decompiler ο οποίος έχει πολύ καλά αποτελέσματα κατά την χρήση του αλλά δυστυχώς παρουσιάζει το μειονέκτημα ότι δεν υποστηρίζει σύγχρονα JDK (το τελευταίο που υποστηρίζει είναι το 1.4). Ύστερα, ο DAVA υστερεί στο ότι κάνει χρήση πολλών δομών δεδομένων και ο αλγόριθμος υλοποίησης του είναι αρκετά περίπλοκος με αποτέλεσμα να χρειάζεται αρκετούς χωρικούς και χρονικούς πόρους. Συγκεκριμένα οι δομές δεδομένων που χρησιμοποιεί προκειμένου να αναπαρασταθεί μια μέθοδος Java είναι πέντε:

- Grimp Representation
- Control Flow Graph (CFG)
- Structure Encapsulation Tree (SET)
- Abstract Syntax Tree (AST)
- Java Source Code

Παρότι ο DAVA δεν λειτουργεί ικανοποιητικά όσον αφορά την παρουσίαση ιδιωμάτων, την εποχή που βγήκε ξεπερνούσε σε απόδοση την πλειοψηφία των decompilers επειδή είναι σε θέση να χειριστεί μια μεγάλη ποικιλία bytecodes. [4]

B'.0.3 Ο decompiler JODE

Ο Jode είναι ένας decompiler ανοιχτού κώδικα ο οποίος παρέχει και έναν optimizer ο οποίος υποστηρίζει τις εξής λειτουργίες: μετονομάζει κλάσεις, πεδία, μεθόδους σε συντομότερα μοναδικά ονόματα, αφαιρεί όλες τις πληροφορίες που σχετίζονται με το debugging, αφαιρεί τυχόν 'νεκρό' κώδικα και βελτιστοποιεί την κατανομή των τοπικών μεταβλητών. Ο decompiler αυτός αντιμετωπίζει κάποια προβλήματα που αφορούν κάποιες συναρτήσεις πρόσβασης της java, της οποίες δεν μπορεί να κατανοήσει ενώ υπάρχουν και περιπτώσεις που παράγει λανθασμένο κώδικα. [5]

B'.0.4 Ο decompiler JAD

Ο JAD είναι ένας decompiler που αναπτύχθηκε σε C++ από τον Pavel Kouznetsov πριν το 1999, ο οποίος δεν συντηρείται πλέον και παρέχει μια διεπαφή με το χρήστη μέσω γραμμής εντολών ώστε να μετατρέψει τα class αρχεία σε πηγαίο κώδικα. Η πιο γνωστή διεπαφή γραφικού περιβάλλοντος για τον JAD είναι ο DJ Java Decompiler. Σύμφωνα με το CNET, ο DJ Java Decompiler έχει χρησιμοποιηθεί πάνω από 10 εκατομμύρια φορές τα προηγούμενα 8 χρόνια. Ένα άλλο γραφικό περιβάλλον για τον JAD είναι το Jadclipse το οποίο είναι και plugin για το Eclipse. Η πιο πρόσφατη έκδοση του JAD υποστηρίζει μόνο τα αρχεία class των εκδόσεων 45.3, 46.0, 47.0 και καθόλου αυτά που παράγονται από την Java 5. [6]

B'.0.5 Ο decompiler JD

Ο decompiler αυτός είναι ένα λογισμικό αρκετά δυνατό και με πολύ καλά αποτελέσματα. Υποστηρίζει αρχεία class που παράγει η Java 5 καθώς και μεταγενέστερες εκδόσεις της γλώσσας. Επιπλέον, υπάρχει ως πρόσθετο plug-in για το περιβάλλον ανάπτυξης προγραμμάτων Java, Eclipse (JD-Eclipse). Εκτός αυτού, αποτελεί και βιβλιοθήκη (JD - Core) ανακατασκευής πηγαίου κώδικα από class αρχεία. Το JD-GUI στα πλαίσια αυτού του λογισμικού είναι ένα γραφικό περιβάλλον, το οποίο ενσωματώνει τη λειτουργία του decompiler ως ξεχωριστή και ανεξάρτητη εφαρμογή. Μέχρι στιγμής δεν υπάρχουν κριτικές ή αναφορές στον τρόπο λειτουργίας του. [7]

B'.0.6 Ο decompiler Mocha

Ο Mocha είναι ένας decompiler ο οποίος αναπτύχθηκε το 1996 από τον Hanpeter van Vliet, μαζί με έναν obfuscator που ονομαζόταν Crema. Λόγω μιας διαμάχης που αφορούσε το γεγονός ότι ένας decompiler θα μπορούσε να βλάψει πολλούς προγραμματιστές αν φανερώνει τον πηγαίο κώδικα τους, ο Hanpeter van Vliet, απέσυρε τον Mocha και σταμάτησε τη διανομή του. Από το 2009, το προγραμμα είναι ξανά διαθέσιμο για ελεύθερη χρήση. Στα μειονεκτήματα του Mocha συγκαταλέγεται το γεγονός ότι η εφαρμογή τερματίζει λόγω αδυναμίας του λογισμικού να συνεχίσει τη λειτουργία του όταν δέχεται σαν είσοδο κάποια αρχεία class και επίσης το γεγονός ότι παρουσιάζει μερικές φορές κάποια όχι και τόσο σημαντικά λάθη τα οποία προέρχονται από τον τρόπο που γίνεται η μεταγλώττιση. Κλασικό παράδειγμα του δεύτερου μειονεκτήματος του Mocha είναι για παράδειγμα το γεγονός ότι κάποιες φορές αδυνατεί να

αναγνωρίσει ότι μια ακέραιη τιμή αναπαριστά μια boolean τιμή και όχι έναν ακέραιο αριθμό. [8]

B'.0.7 Ο decompiler DeJaVu

Ο DeJaVu είναι ένας ακόμα decompiler ο οποίος έχει τα πλεονεκτήματα ότι δεν παράγει λάθη οφειλόμενα στη μεταγλώττιση τα οποία δεν διορθώνονται εύκολα (όπως για παράδειγμα κώδικας που να περιέχει εντολές goto), δεν εμφανίζει λάθη τα οποία να οδηγούν σε απρόσμενο τερματισμό της εφαρμογής και δεν εμφανίζει λάθη συντακτικά (όπως για παράδειγμα μη χρήση του this). Παρόλα αυτά ο decompiler αυτός έχει σημαντικά προβλήματα που αφορούν τη διαχείριση της ροής, παράγει ελαφρώς λανθασμένο κώδικα και παράγει και ορισμένα λάθη τα οποία είναι εύκολο να διορθωθούν όπως για παράδειγμα το γεγονός ότι κάποιες φορές αδυνατεί να αναγνωρίσει ότι μια ακέραιη τιμή αναπαριστά μια boolean τιμή και όχι έναν ακέραιο αριθμό. [8]

B'.0.8 Ο decompiler WingDis

Ο WingDis είναι ένας εμπορικός decompiler ο οποίος δεν εμφανίζει λάθη οφειλόμενα στη μεταγλώττιση τα οποία δεν διορθώνονται εύκολα, ούτε εμφανίζει λάθη τα οποία οδηγούν στον τερματισμό της εφαρμογής ή δημιουργούν ελαφρώς λανθασμένο κώδικα. Τα κλασικά λάθη που παράγει είναι όμως η υπερβολική χρήση του if(x!=false) και άλλων παρόμοιων δομών καθώς και η κακή χρήση ή η καθόλου χρήση του super αλλά και η άστοχη μετάφραση του x=a++ σε a++; x=a; [8]

Άλλοι decompilers που έχουν αναπτυχθεί, δεν συντηρούνται πλέον (έχουν σταματήσει στην έκδοση java 2) και δεν υπάρχουν αρκετά στοιχεία για την απόδοσή τους είναι οι: JREVERSE PRO, JDEC, JASCI και SOURCEDEC. [9]

B'.0.9 Ο decompiler CFR

Ο CFR είναι ένας Java Decompiler ο οποίος είναι γνωστός για την ακρίβειά του στην διαδικασία του decompiling. Ενημερώνεται συχνά έτσι ώστε να είναι σε θέση να υποστηρίξει τα χαρακτηριστικά που προστίθενται στις νέες εκδόσεις της γλώσσας Java. [13]

B'.0.10 O decompiler Procyon

Ο Procyon είναι ένας Java Decompiler ανοιχτού κώδικα ο οποίος προσφέρει έναν αξιόπιστο τρόπο για να αποκτήσει κανείς τον πηγαίο κώδικα που αντιστοιχεί σε ένα class αρχείο. Υποστηρίζει αρκετά νέα χαρακτηριστικά της γλώσσας Java όπως είναι οι εκφράσεις Lamda, οι εμφωλευμένες κλάσεις, τα annotations και τα enumerations. [14]

B'.0.11 O decompiler FernFlower

Ο Fernflower είναι ένας Java Decompiler ο οποίος έχει αναπτυχθεί από τους δημιουργούς του ολοκληρωμένου προγραμματιστικού περιβάλλοντος IntelliJ και ενημερώνεται μέχρι και σήμερα. Είναι γνωστός για την ακρίβεια του στην διαδικασία του decompiling περίπλοκου κώδικα Java. [15]

Κεφάλαιο Γ΄

ΤΟ ΣΥΝΤΑΚΤΙΚΟ ΔΕΝΤΡΟ

Γ΄.1 Το συντακτικό δέντρο

Το συντακτικό δέντρο αποτελεί την κύρια δομή που χρησιμοποιείται από τον decompiler. Κατά τη διαδικασία ανάλυσης των εντολών bytecode χτίζεται σταδιακά το συντακτικό δέντρο το οποίο περιέχει στους κόμβους του τον αντίστοιχο πηγαίο κώδικα. Οι κανόνες με βάση τους οποίους αναπαρίσταται το συντακτικό δέντρο είναι οι εξής:

- Κάθε κόμβος αναπαριστά μια εντολή ή το τέλος μιας εντολής.
- Κάθε κόμβος έχει ως δύο παιδιά.
- Το αριστερό παιδί συνδέεται με τον γονέα του με ένα πράσινο βέλος.
- Το δεξί παιδί συνδέεται με τον γονέα του με ένα κόκκινο βέλος.
- Αν ένας κόμβος δεν έχει αριστερό ή δεξί παιδί, ή δεν έχει κανένα παιδί, τότε αυτοί οι κόμβοι - παιδιά δεν αναπαρίστανται στο δέντρο.
- Κάθε κόμβος μπορεί να έχει παραπάνω από έναν γονέα.

Γ΄.2 Παραδείγματα συντακτικών δέντρων

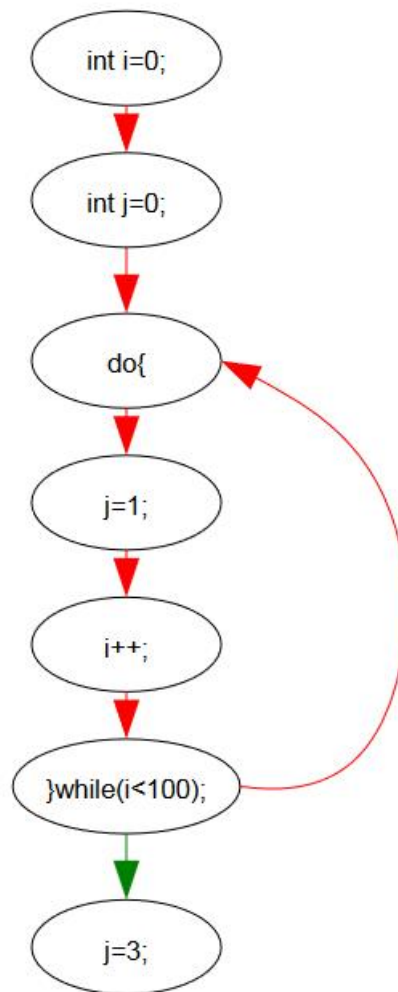
Επαναληπτική δομή do while

Πηγαίος Κώδικας:

```
public class dowhile{
    public void spin() {
        int i = 0;
        int j;
        do{
            j=1;
            i+=1;
        }while(i < 100);
        j=3;
    }
}
```

Συντακτικό Δέντρο:

Στην δομή do while όπως βλέπουμε στο παρακάτω συντακτικό δέντρο, ο κόμβος με την συνθήκη ελέγχου έχει δύο παιδιά. Στη δομή do while όταν η συνθήκη ελέγχου είναι αληθής, η εκτέλεση του προγράμματος συνεχίζεται από την εντολή που αντιστοιχεί στην αρχή του βρόγχου επανάληψης ενώ όταν η συνθήκη ελέγχου είναι ψευδής συνεχίζεται από την εντολή που έπεται της συνθήκης ελέγχου. Επομένως το δεξί παιδί του κόμβου που περιέχει την συνθήκη ελέγχου του βρόγχου επανάληψης, είναι ο κόμβος με την εντολή 'do{'. Στο σχήμα βλέπουμε το κόκκινο βέλος το οποίο πάει προς τα πίσω και συνδέει τον κόμβο (γονέα) που περιέχει την συνθήκη ελέγχου του βρόγχου (}while(i<100);) με τον κόμβο (δεξί παιδί) που περιέχει την εντολή της εκκίνησης του βρόγχου '(do{)'. Το αριστερό παιδί του κόμβου της συνθήκης ελέγχου είναι ο κόμβος με την εντολή 'j=3;'. Στο σχήμα βλέπουμε το πράσινο βέλος το οποίο συνδέει τον κόμβο (γονέα) που περιέχει την συνθήκη ελέγχου του βρόγχου, με τον κόμβο (αριστερό παιδί) που περιέχει την εντολή που έπεται του βρόγχου (j=3;).



Εμφωλευμένες δομές do while

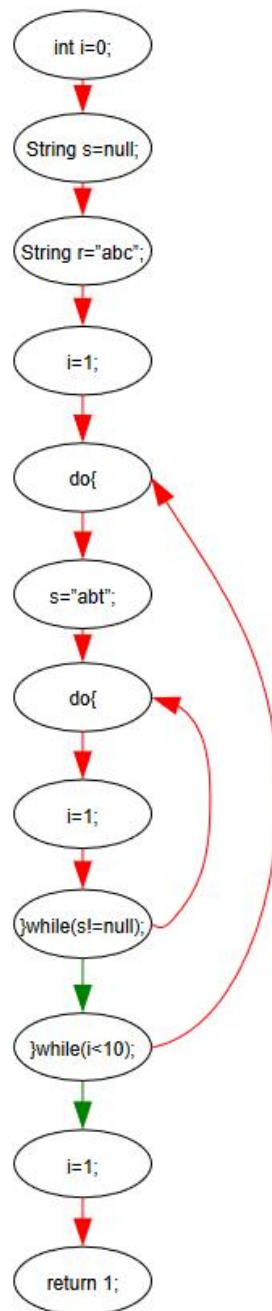
Πηγαίος Κώδικας:

```
public class nesteddowhiles2{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
```

```
        i=1;
        do{
            s = "abt";
            do{
                i= 1;
            } while (s!=null );
        } while (i <10);
        i= 1;
        return 1;
    }
}
```

Συντακτικό Δέντρο:

Στο παρακάτω σχήμα βλέπουμε το συντακτικό δέντρο που δημιουργείται όταν έχουμε εμφωλευμένες επαναληπτικές δομές do while. Η διαδικασία δημιουργίας του είναι ίδια με αυτήν που ήδη αναφέρθηκε στην περίπτωση της ύπαρξης μιας μόνο επαναληπτικής δομής do-while.

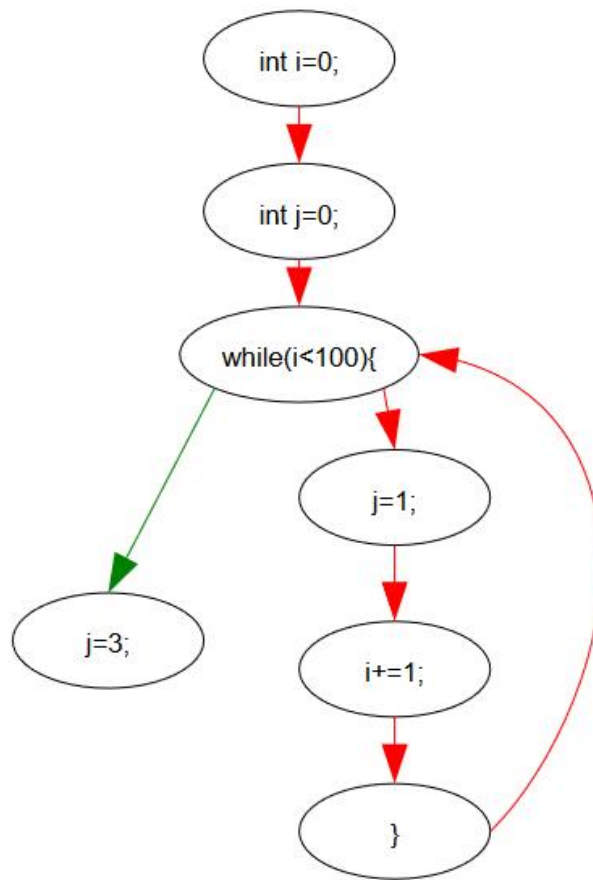


Επαναληπτική δομή while**Πηγαίος Κώδικας:**

```
public class whiletest{
    public void spin() {
        int i = 0;
        int j = 0;
        while(i < 100) {
            j=1;
            i+=1;
        }
        j = 3;
    }
}
```

Συντακτικό Δέντρο:

Στην δομή while όπως βλέπουμε στο παρακάτω συντακτικό δέντρο, ο κόμβος με την συνθήκη ελέγχου έχει δύο παιδιά. Στη δομή while όταν η συνθήκη ελέγχου είναι αληθής, εκτελούνται οι εντολές που βρίσκονται εντός του βρόγχου και στη συνέχεια η εκτέλεση του προγράμματος συνεχίζεται από την εντολή που αντιστοιχεί στην αρχή του βρόγχου επανάληψης η οποία περιέχει την συνθήκη ελέγχου. Όταν η συνθήκη ελέγχου είναι ψευδής η εκτέλεση συνεχίζεται από την εντολή που έπεται του βρόγχου. Επομένως το δεξί παιδί του κόμβου που περιέχει την συνθήκη ελέγχου του βρόγχου επανάληψης, είναι ο κόμβος που περιέχει την πρώτη εντολή που βρίσκεται εντός του βρόγχου ('j=1;'). Το δεξί παιδί του κόμβου που κλείνει τον βρόγχο επανάληψης (περιέχει το σύμβολο '}'), είναι ο κόμβος που περιέχει την συνθήκη ελέγχου. Στο σχήμα βλέπουμε το κόκκινο βέλος το οποίο πάει προς τα πίσω και συνδέει τον κόμβο (γονέα) που περιέχει το σύμβολο τερματισμού του βρόγχου (}) με τον κόμβο (δεξί παιδί) που περιέχει την συνθήκη ελέγχου του βρόγχου (while(i<100)). Το αριστερό παιδί του κόμβου της συνθήκης ελέγχου είναι ο κόμβος με την εντολή 'j=3;'. Στο σχήμα βλέπουμε το πράσινο βέλος το οποίο συνδέει τον κόμβο (γονέα) που περιέχει την συνθήκη ελέγχου του βρόγχου, με τον κόμβο (αριστερό παιδί) που περιέχει την εντολή που έπεται του βρόγχου (j=3;).



Εμφωλευμένες δομές while

Πηγαίος Κώδικας:

```

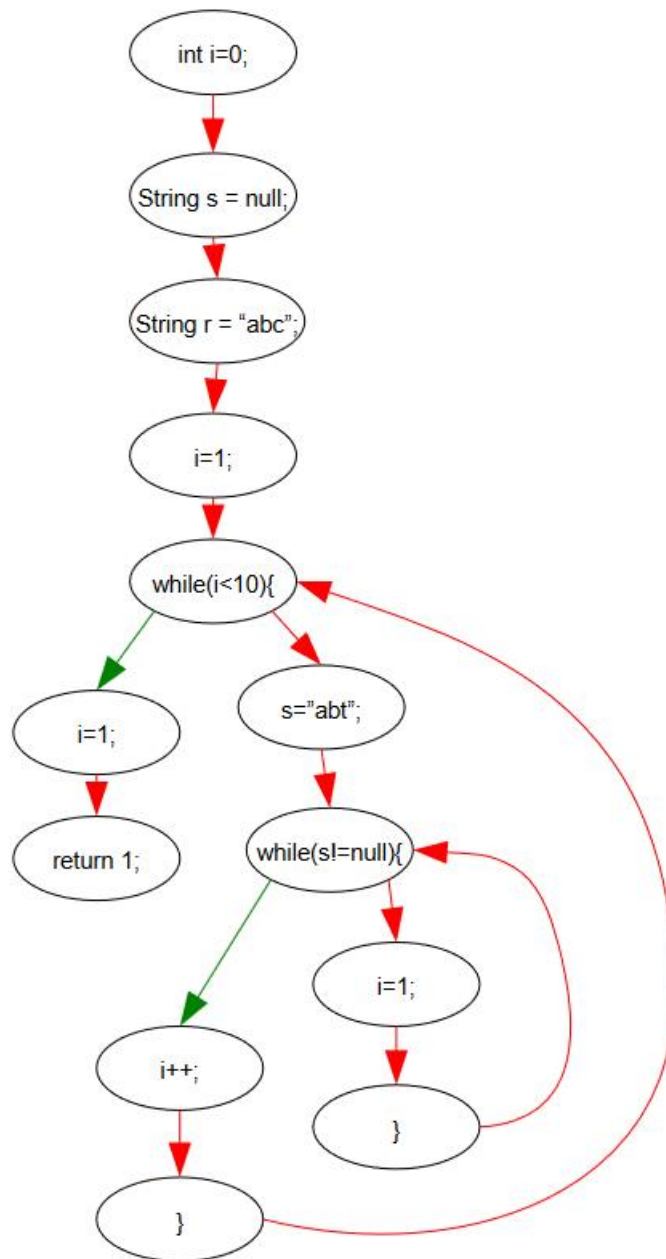
public class nestedwhiles{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        i=1;
        while(i<10){
            s = "abt";
        }
    }
}
  
```

```
        while (s!=null){
            i= 1;

        }
        i++;
    }
    i= 1;
    return 1;
}
}
```

Συντακτικό Δέντρο:

Στο παρακάτω σχήμα βλέπουμε το συντακτικό δέντρο που δημιουργείται όταν έχουμε εμφωλευμένες επαναληπτικές δομές while. Η διαδικασία δημιουργίας του είναι ίδια με αυτήν που ήδη αναφέρθηκε στην περίπτωση της ύπαρξης μιας μόνο επαναληπτικής δομής do-while.



Δομή υπόθεσης if**Πηγαίος Κώδικας:**

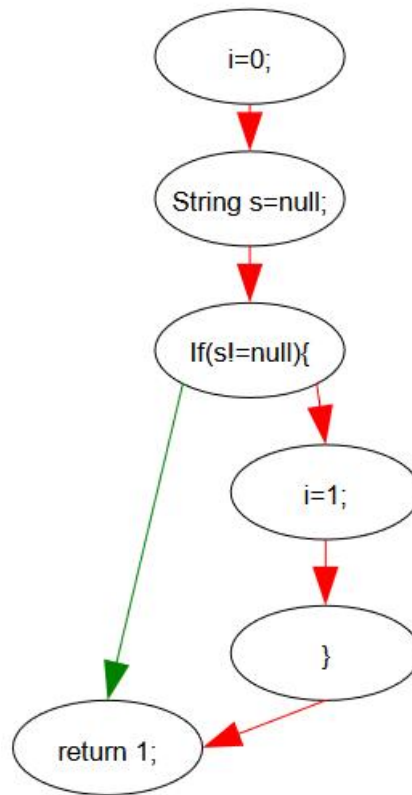
```

public class test3{
    public int add(){
        int i=0;
        String s=null;
        if(s!=null){
            i= 1;
        }
        return 1;
    }
}

```

Συντακτικό Δέντρο:

Στην δομή if όπως βλέπουμε στο παρακάτω συντακτικό δέντρο, ο κόμβος με την συνθήκη ελέγχου έχει δύο παιδιά. Στη δομή if όταν η συνθήκη ελέγχου είναι αληθής, εκτελούνται οι εντολές που βρίσκονται εντός του block υπόθεσης και στη συνέχεια η εκτέλεση του προγράμματος συνεχίζεται από την εντολή που έπεται του block υπόθεσης. Όταν η συνθήκη ελέγχου είναι ψευδής, οι εντολές που βρίσκονται εντός του block υπόθεσης παραλείπονται και η εκτέλεση συνεχίζεται από την εντολή που έπεται του block υπόθεσης. Επομένως το δεξί παιδί του κόμβου που περιέχει την συνθήκη ελέγχου, είναι ο κόμβος που περιέχει την πρώτη εντολή που βρίσκεται εντός του block υπόθεσης ('i=1;'). Το δεξί παιδί του κόμβου που κλείνει το block υπόθεσης (περιέχει το σύμβολο '}'), είναι ο κόμβος που περιέχει την εντολή που έπεται του block υπόθεσης. Στο σχήμα βλέπουμε το κόκκινο βέλος το οποίο πάει προς τα εμπρός και συνδέει τον κόμβο (γονέα) που περιέχει το σύμβολο τερματισμού του block υπόθεσης (}) με τον κόμβο (δεξί παιδί) που περιέχει την εντολή που έπεται του block υπόθεσης (return 1;). Το αριστερό παιδί του κόμβου της συνθήκης ελέγχου είναι επίσης ο κόμβος με την εντολή 'return 1;'. Στο σχήμα βλέπουμε το πράσινο βέλος το οποίο συνδέει τον κόμβο (γονέα) που περιέχει την συνθήκη ελέγχου, με τον κόμβο (αριστερό παιδί) που περιέχει την εντολή που έπεται του block υπόθεσης (return 1;).

Δομή υπόθεσης if - else**Πηγαίος Κώδικας:**

```
public class test{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        if (s!=null){
            i= 1;
        }
        else{
            i = 0;
        }
    }
}
```

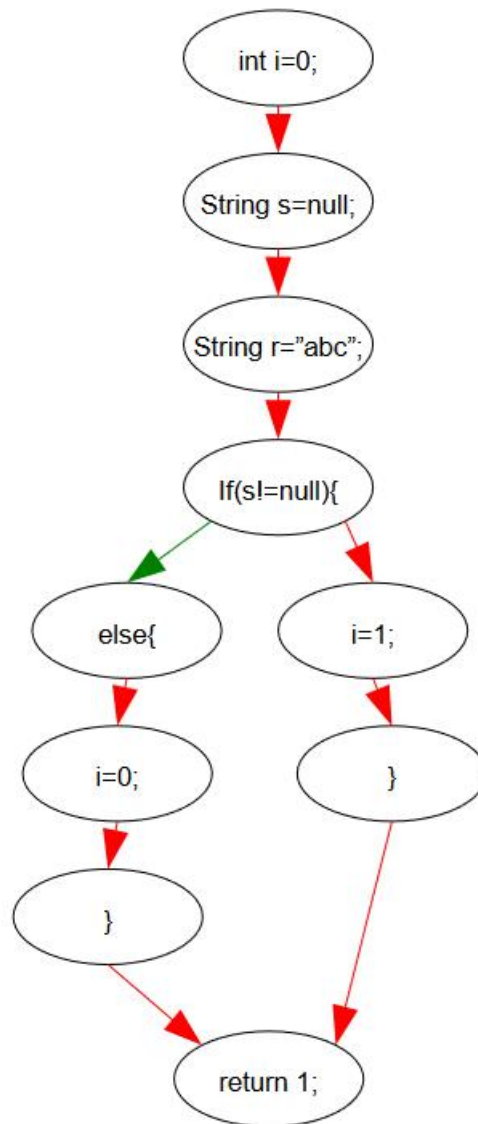
```

    }
    return 1;
}
}

```

Συντακτικό Δέντρο:

Στην δομή if - else όπως βλέπουμε στο παρακάτω συντακτικό δέντρο, ο κόμβος με την συνθήκη ελέγχου έχει δύο παιδιά. Στη δομή if - else όταν η συνθήκη ελέγχου είναι αληθής, εκτελούνται οι εντολές που βρίσκονται εντός του block if και στη συνέχεια οι εντολές που βρίσκονται εντός του block else παραλείπονται και η εκτέλεση συνεχίζεται από την εντολή που έπεται της δομής υπόθεσης. Όταν η συνθήκη ελέγχου είναι ψευδής, η εκτέλεση του προγράμματος συνεχίζεται από την πρώτη εντολή που περιλαμβάνεται στο block else. Επομένως το δεξί παιδί του κόμβου που περιέχει την συνθήκη ελέγχου, είναι ο κόμβος που περιέχει την πρώτη εντολή που βρίσκεται εντός του block if ('i=1;'). Το δεξί παιδί του κόμβου που κλείνει το block if (περιέχει το σύμβολο '}'), είναι ο κόμβος που περιέχει την εντολή που έπεται της δομής υπόθεσης. Στο σχήμα βλέπουμε το κόκκινο βέλος το οποίο πάει προς τα εμπρός και συνδέει τον κόμβο (γονέα) που περιέχει το σύμβολο τερματισμού του block if (}) με τον κόμβο (δεξί παιδί) που περιέχει την εντολή που έπεται της δομής υπόθεσης (return 1;). Το αριστερό παιδί του κόμβου της συνθήκης ελέγχου είναι ο κόμβος με την εντολή 'else{'. Στο σχήμα βλέπουμε το πράσινο βέλος το οποίο συνδέει τον κόμβο (γονέα) που περιέχει την συνθήκη ελέγχου, με τον κόμβο (αριστερό παιδί) που περιέχει την εντολή που έπεται του block if, την εντολή (else{). Ο κόμβος με την εντολή else έχει ένα μόνο παιδί το δεξί, το οποίο είναι ο κόμβος ο οποίος περιέχει την πρώτη εντολή που περιλαμβάνεται εντός του block else ('i=0;'). Ο κόμβος που περιέχει το σύμβολο τερματισμού του block else έχει, όπως βλέπουμε και στο σχήμα με το κόκκινο βέλος, δεξί παιδί επίσης την εντολή που έπεται της δομής υπόθεσης η οποία είναι η return 1;.

Δομή υπόθεσης if - else if - else**Πηγαίος Κώδικας:**

```
public class test4{  
    public int add(){  
        int i=3;
```

```

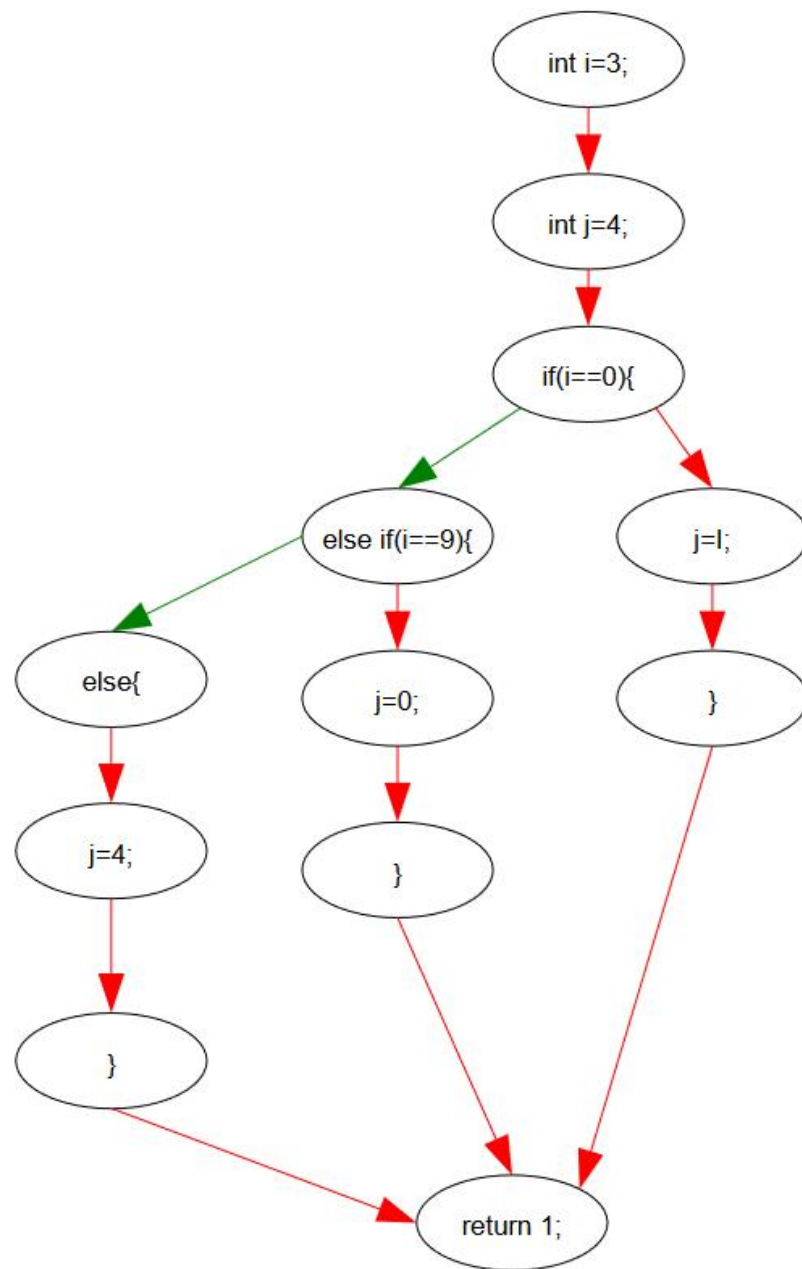
        int j=4;
        if ( i==0){
            j=1;
        }
        else if ( i==9){
            j=0;
        }
        else {
            j=4;
        }
        return 1;
    }
}

```

Συντακτικό Δέντρο:

Στην δομή if - else if - else όπως βλέπουμε στο παρακάτω συντακτικό δέντρο, ο κόμβος με την συνθήκη ελέγχου της εντολής if έχει δύο παιδιά. Στη δομή if - else if - else όταν η συνθήκη ελέγχου της εντολής if είναι αληθής, εκτελούνται οι εντολές που βρίσκονται εντός του block if. Στη συνέχεια οι εντολές που βρίσκονται εντός των block else if και else παραλείπονται και η εκτέλεση συνεχίζεται από την εντολή που έπεται της δομής υπόθεσης. Αν η συνθήκη ελέγχου είναι ψευδής, η εκτέλεση του προγράμματος συνεχίζεται από την συνθήκη ελέγχου της εντολής else if. Επομένως το δεξί παιδί του κόμβου που περιέχει την συνθήκη ελέγχου if, είναι ο κόμβος που περιέχει την πρώτη εντολή που βρίσκεται εντός του block if ('j=1;'). Το δεξί παιδί του κόμβου που κλείνει το block if (περιέχει το σύμβολο '}'), είναι ο κόμβος που περιέχει την εντολή που έπεται της δομής υπόθεσης. Στο σχήμα βλέπουμε το κόκκινο βέλος το οποίο πάει προς τα εμπρός και συνδέει τον κόμβο (γονέα) που περιέχει το σύμβολο τερματισμού του block if (}) με τον κόμβο (δεξί παιδί) που περιέχει την εντολή που έπεται της δομής υπόθεσης (return 1;). Το αριστερό παιδί του κόμβου της συνθήκης ελέγχου if είναι ο κόμβος με την συνθήκη ελέγχου της εντολής 'else if'. Στο σχήμα βλέπουμε το πράσινο βέλος το οποίο συνδέει τον κόμβο (γονέα) που περιέχει την συνθήκη ελέγχου if, με τον κόμβο (αριστερό παιδί) που περιέχει την εντολή που έπεται του block if, την συνθήκη ελέγχου της εντολής (else if). Όταν η συνθήκη ελέγχου της εντολής else if είναι αληθής, εκτελούνται οι εντολές που βρίσκονται εντός του block else if και στη συνέχεια οι εντολές που βρίσκονται εντός του block else παραλείπονται και η εκτέλεση συνεχίζεται από την εντολή που έπεται

της δομής υπόθεσης. Όταν η συνθήκη ελέγχου είναι ψευδής, η εκτέλεση του προγράμματος συνεχίζεται από την πρώτη εντολή που περιλαμβάνεται στο block else. Επομένως το δεξί παιδί του κόμβου που περιέχει την συνθήκη ελέγχου else if, είναι ο κόμβος που περιέχει την πρώτη εντολή που βρίσκεται εντός του block else ('j=0;'). Το δεξί παιδί του κόμβου που κλείνει το block else if (περιέχει το σύμβολο '}'), είναι ο κόμβος που περιέχει την εντολή που έπεται της δομής υπόθεσης. Στο σχήμα βλέπουμε το κόκκινο βέλος το οποίο πάει προς τα εμπρός και συνδέει τον κόμβο (γονέα) που περιέχει το σύμβολο τερματισμού του block else if (}) με τον κόμβο (δεξί παιδί) που περιέχει την εντολή που έπεται της δομής υπόθεσης (return 1;). Το αριστερό παιδί του κόμβου της συνθήκης ελέγχου else if είναι ο κόμβος με την εντολή 'else'. Ο κόμβος με την εντολή else έχει ένα μόνο παιδί το δεξί, το οποίο είναι ο κόμβος ο οποίος περιέχει την πρώτη εντολή που περιλαμβάνεται εντός του block else ('j=4;'). Ο κόμβος που περιέχει το σύμβολο τερματισμού του block else έχει, όπως βλέπουμε και στο σχήμα με το κόκκινο βέλος, δεξί παιδί επίσης την εντολή που έπεται της δομής υπόθεσης η οποία είναι η return 1;.



Εμφωλευμένες δομές υπόθεσης

Στα παρακάτω σχήματα βλέπουμε τα συντακτικά δέντρα που δημιουργούνται όταν έχουμε εμφωλευμένες δομές υπόθεσης. Η διαδικασία δημιουργίας τους είναι ίδια με αυτήν που ήδη αναφέρθηκε στην περίπτωση της ύπαρξης μιας μόνο δομής υπόθεσης.

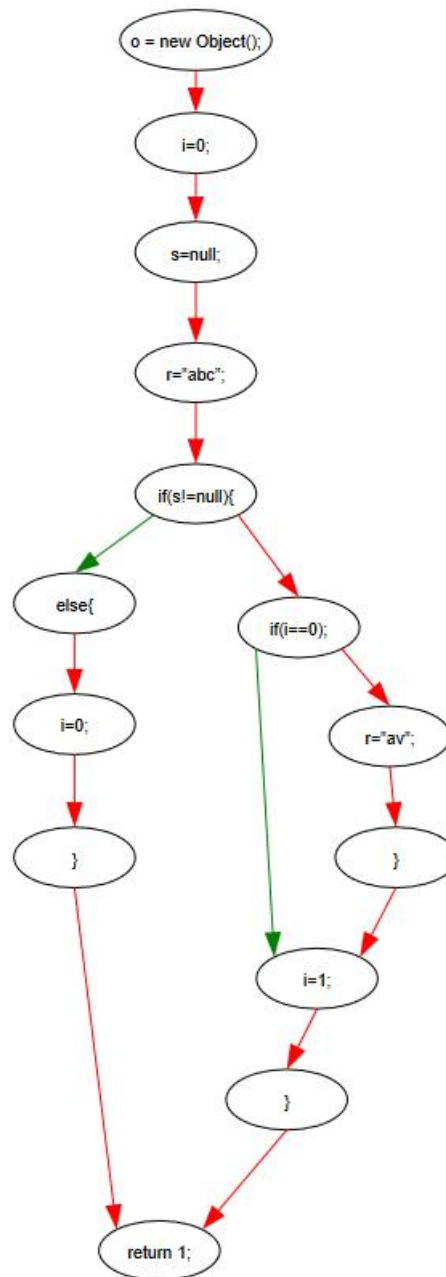
Παράδειγμα 1**Πηγαίος Κώδικας:**

```
public class nestedifs2{

    public nestedifs2(){
        super();
    }

    public int add(){
        Object o = new Object();
        int i=0;
        String s=null;
        String r = "abc";
        if(s!=null){
            if(i==0){
                r = "av";
            }
            i= 1;
        }
        else{
            i = 0;
        }
        return 1;
    }
}
```

Συνταχτικό Δέντρο:



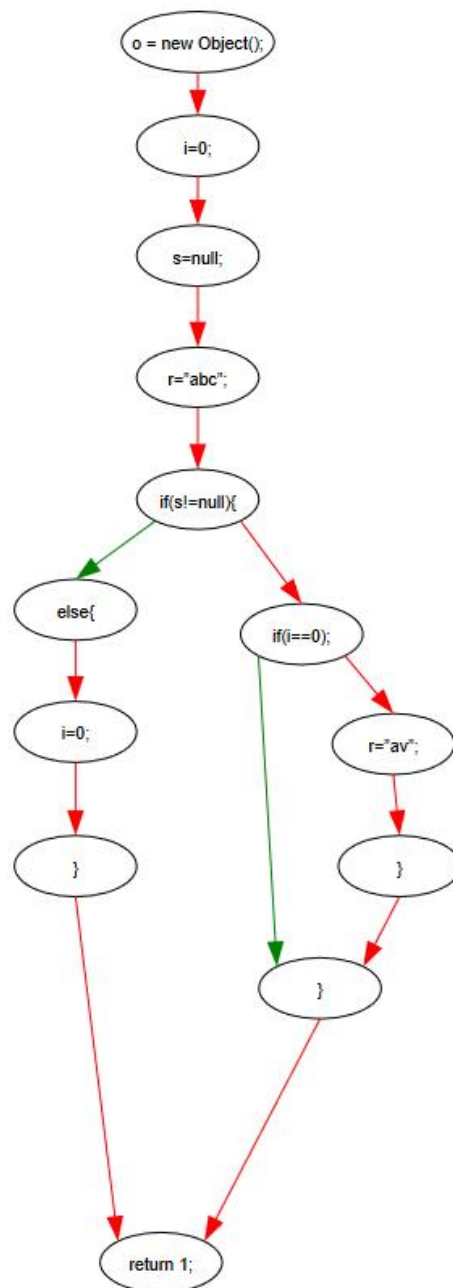
Παράδειγμα 2**Πηγαίος Κώδικας:**

```
public class nestedifs22{

    public nestedifs22(){
        super();
    }

    public int add(){
        Object o = new Object();
        int i=0;
        String s=null;
        String r = "abc";
        if(s!=null){
            if(i==0){
                r = "av";
            }
        }
        else{
            i = 0;
        }
        return 1;
    }
}
```

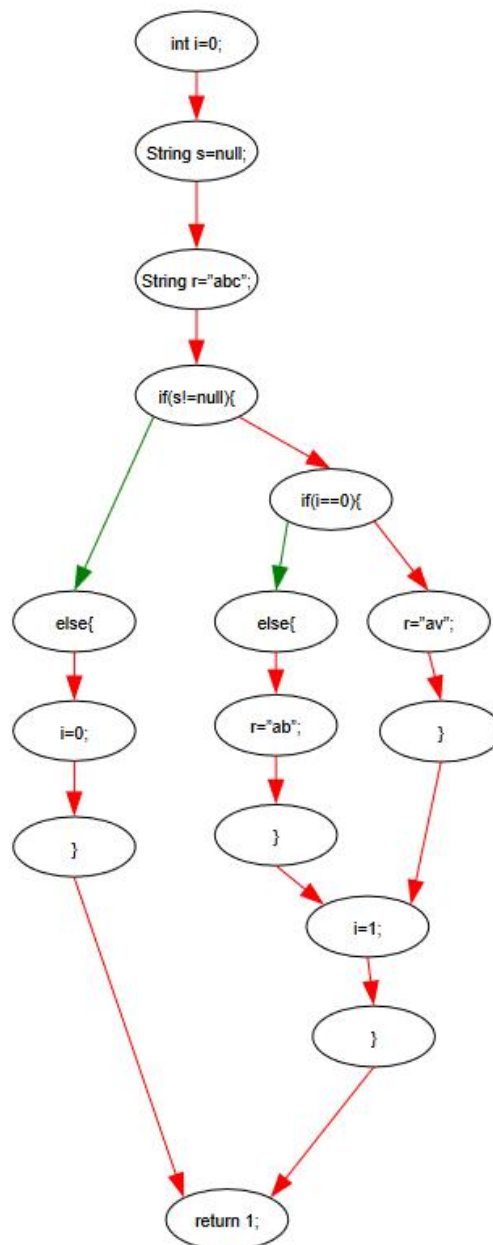
Συντακτικό Δέντρο:



Παράδειγμα 3**Πηγαίος Κώδικας:**

```
public class nestedifs3{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        if(s!=null){
            if(i==0){
                r = "av";
            }
            else{
                r = "ab";
            }
            i= 1;
        }
        else{
            i = 0;
        }
        return 1;
    }
}
```

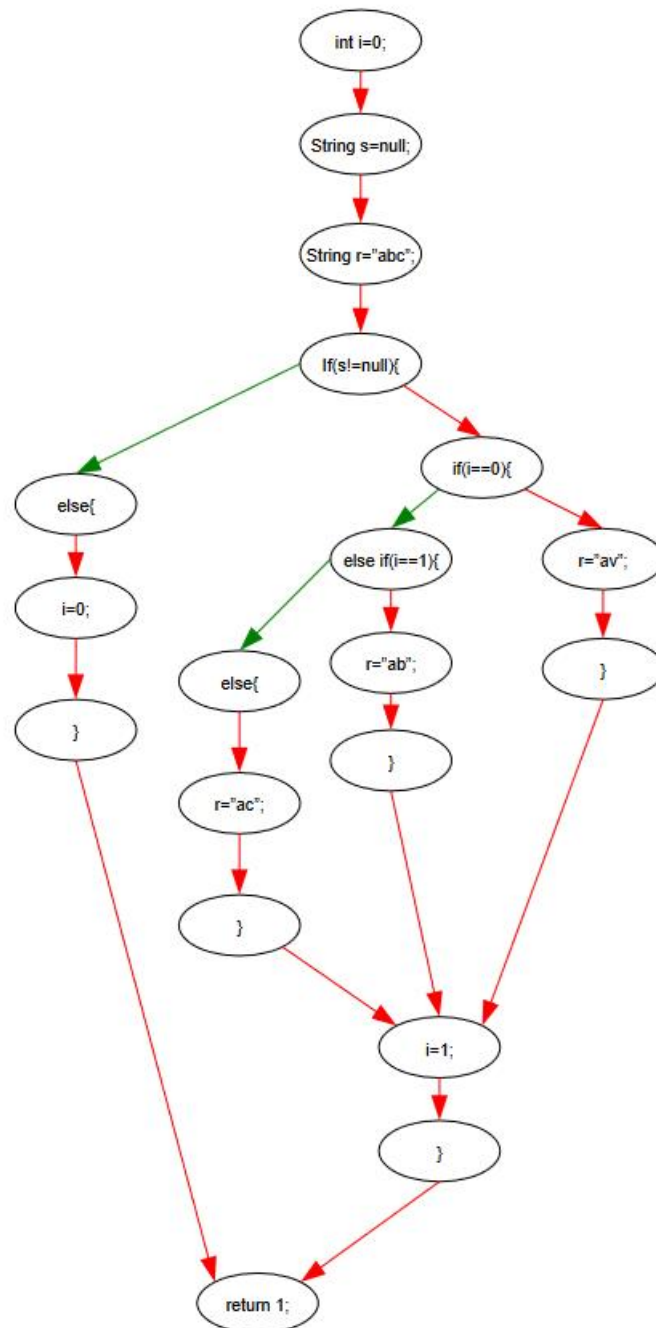
Συντακτικό Δέντρο:



Παράδειγμα 4**Πηγαίος Κώδικας:**

```
public class nestedifs{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        if(s!=null){
            if(i==0){
                r = "av";
            }
            else if(i == 1){
                r = "ab";
            }
            else{
                r = "ac";
            }
            i= 1;
        }
        else{
            i = 0;
        }
        return 1;
    }
}
```

Συνταχτικό Δέντρο:



Εμφωλευμένες δομές

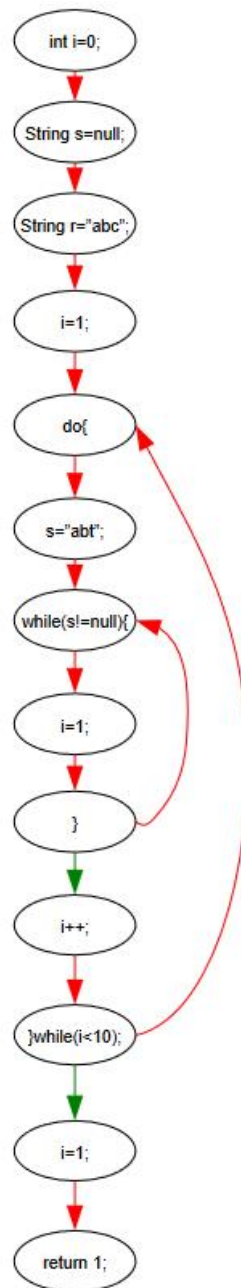
Στα παρακάτω σχήματα βλέπουμε τα συντακτικά δέντρα που δημιουργούνται όταν έχουμε εμφωλευμένες δομές διαφόρων ειδών. Η διαδικασία δημιουργίας τους είναι ίδια με αυτήν που ήδη αναφέρθηκε στην περίπτωση της ύπαρξης μιας μόνο δομής κάθε είδους.

Παράδειγμα 1: Δομή do while με εμφωλευμένη δομή while

Πηγαίος Κώδικας:

```
public class dowhilewhile{
    public int add(){
        int i=0;
        String s=null;
        String r="abc";
        i=1;
        do{
            s="abt";
            while(s!=null){
                i= 1;
            }
            i++;
        }while(i<10);
        i=1;
        return 1;
    }
}
```

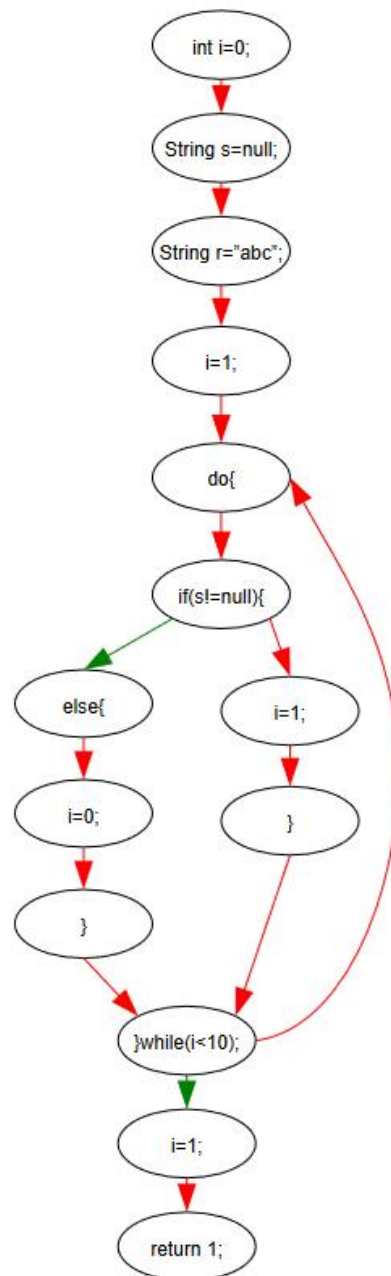
Συνταχτικό Δέντρο:



Παράδειγμα 2: Δομή do while με εμφωλευμένη δομή if - else**Πηγαίος Κώδικας:**

```
public class dowhileif2{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        i=1;
        do{
            if(s!=null){
                i= 1;
            }
            else{
                i = 0;
            }
        }while(i<10);
        i= 1;
        return 1;
    }
}
```

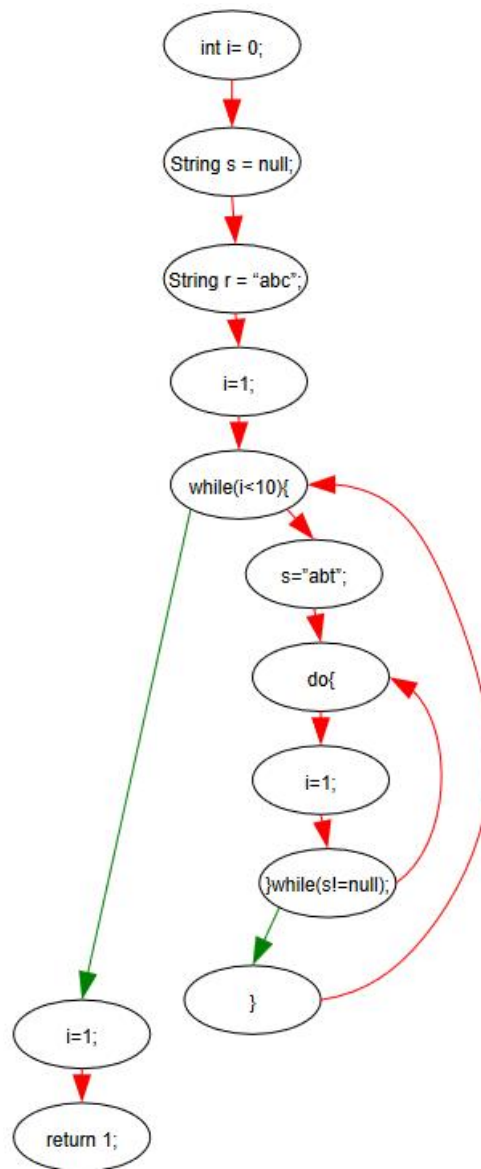
Συντακτικό Δέντρο:



Παράδειγμα 3: Δομή while με εμφωλευμένη δομή do while**Πηγαίος Κώδικας:**

```
public class whiledowhile2{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        i=1;
        while(i<10){
            s="abt";
            do{
                i=1;
            }while(s!=null);
        }
        i=1;
        return 1;
    }
}
```

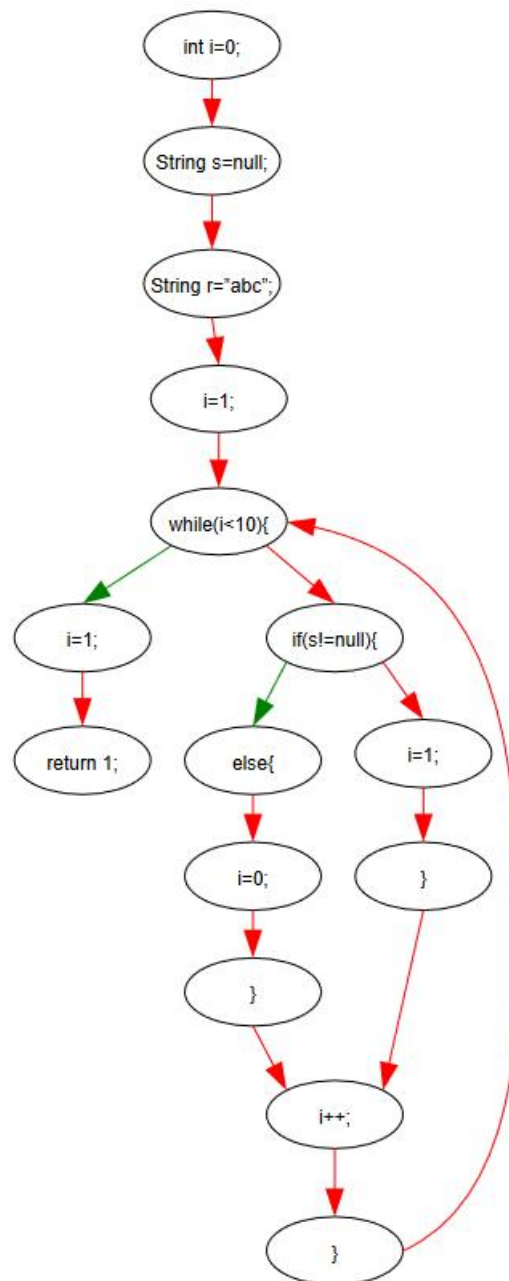
Συνταχτικό Δέντρο:



Παράδειγμα 4: Δομή while με εμφωλευμένη δομή if-else**Πηγαίος Κώδικας:**

```
public class whileif{
    public int add(){
        int i=0;
        String s=null;
        String r="abc";
        i=1;
        while(i<10){
            if(s!=null){
                i=1;
            }
            else{
                i=0;
            }
            i++;
        }
        i=1;
        return 1;
    }
}
```

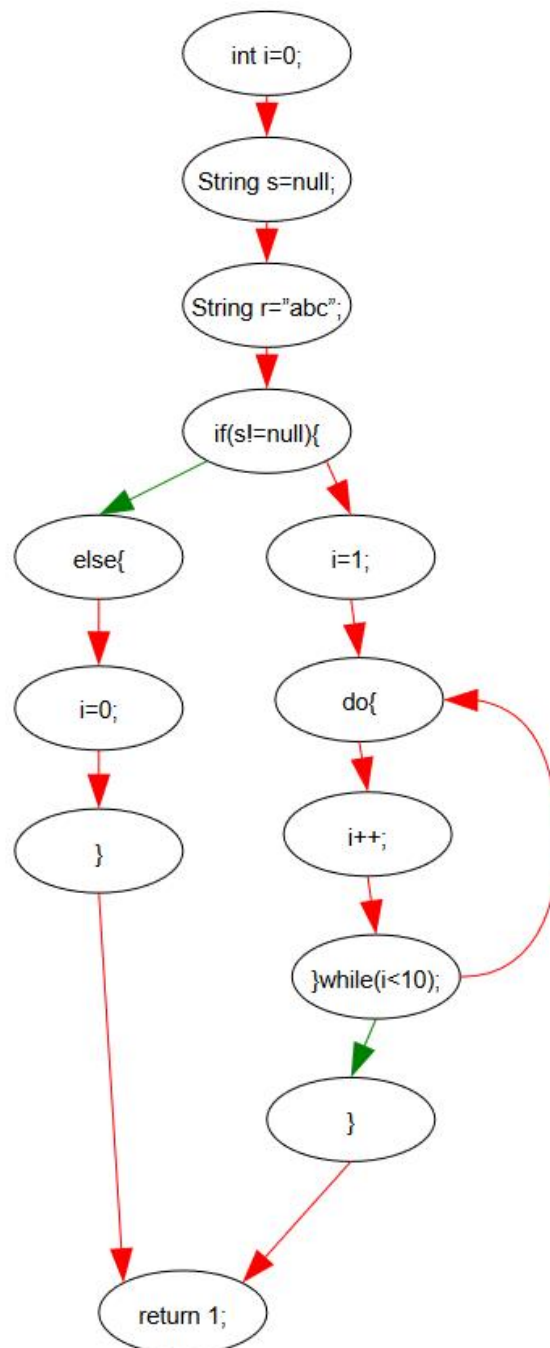
Συντακτικό Δέντρο:



Παράδειγμα 5: Δομή if-else με εμφωλευμένη δομή do while**Πηγαίος Κώδικας:**

```
public class ifdowhile2{
    public int add(){
        int i=0;
        String s=null;
        String r="abc";
        if (s!=null){
            i=1;
            do{
                i++;
            } while (i < 10);
        }
        else {
            i=0;
        }
        return 1;
    }
}
```

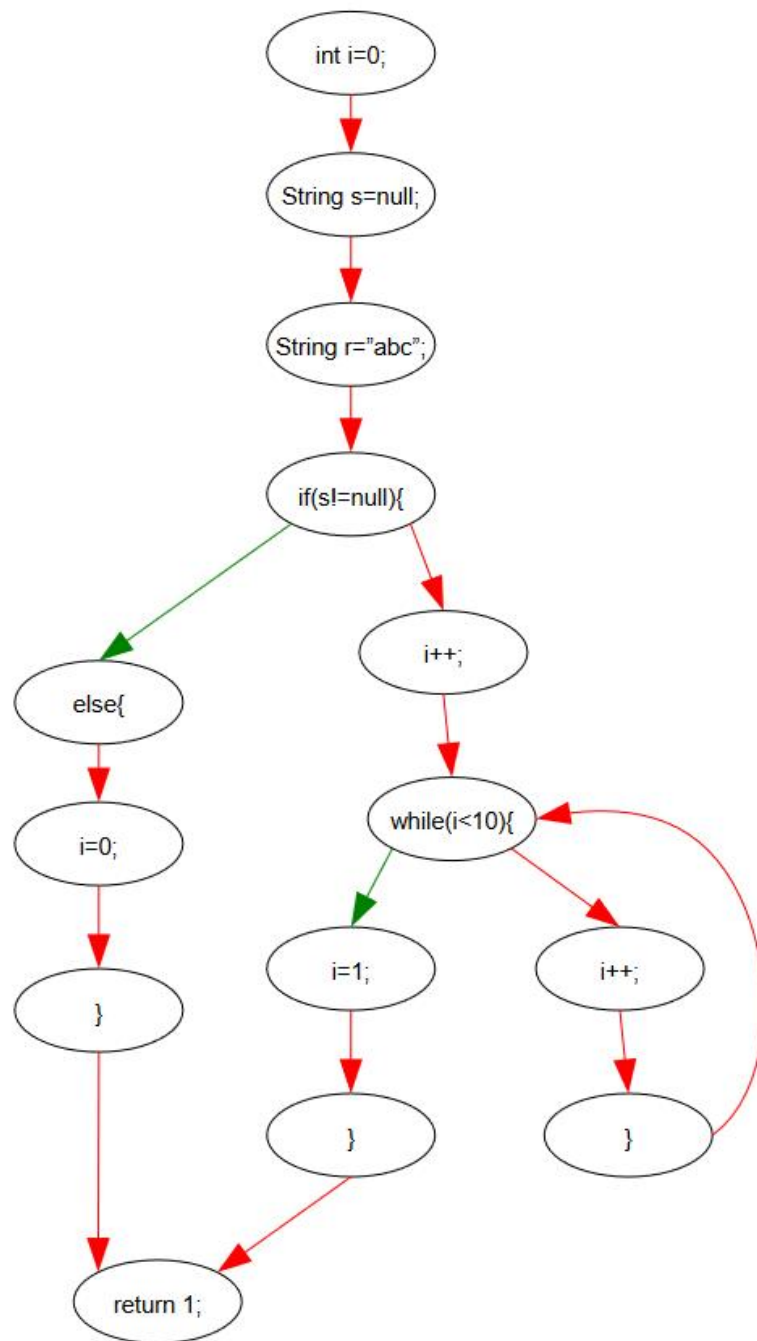
Συνταχτικό Δέντρο:



Παράδειγμα 6: Δομή if-else με εμφωλευμένη δομή while**Πηγαίος Κώδικας:**

```
public class ifwhile{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        if (s!=null){
            i++;
            while(i<10){
                i++;
            }
            i= 1;
        }
        else{
            i = 0;
        }
        return 1;
    }
}
```

Συνταχτικό Δέντρο:

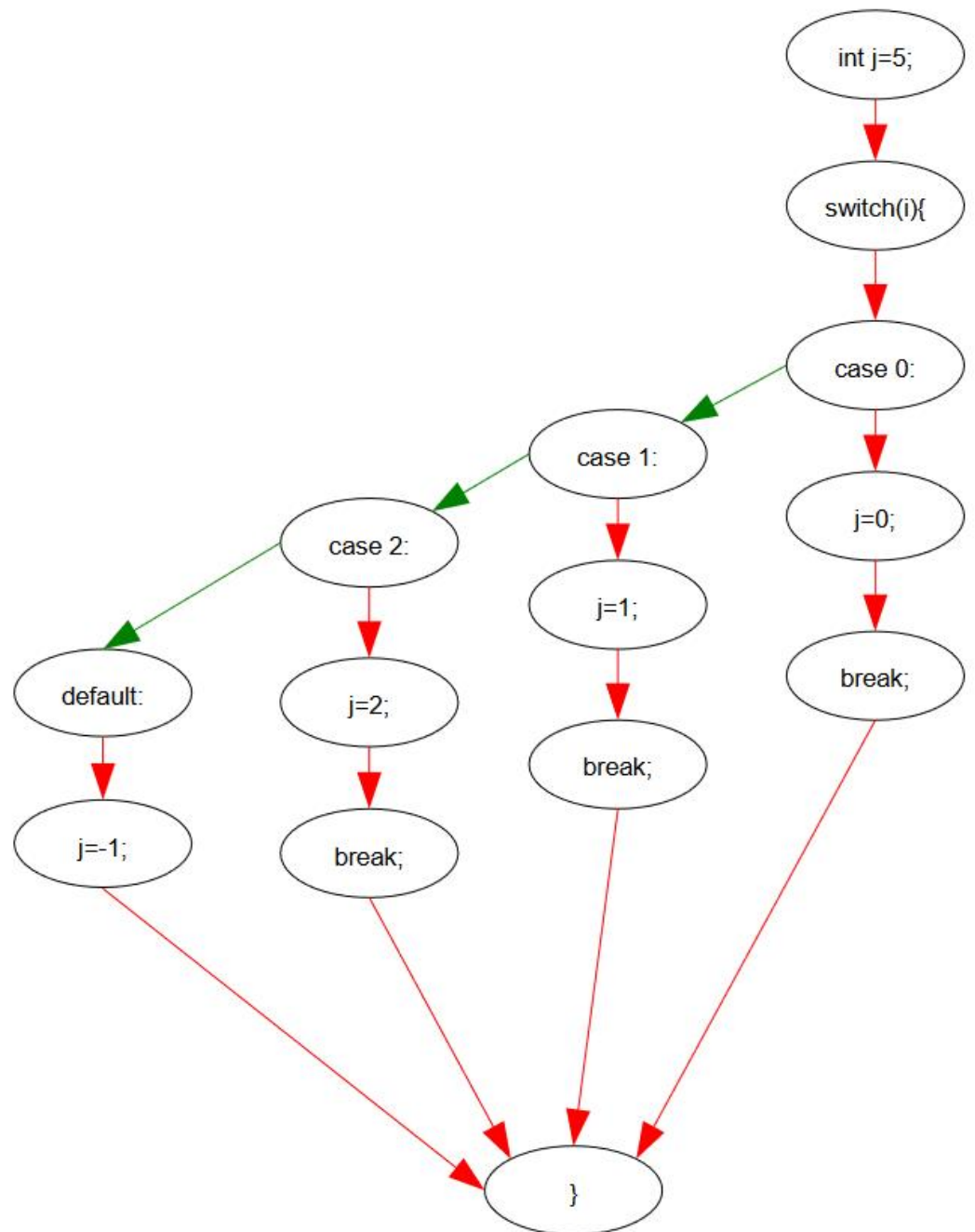


Δομή επιλογής switch**Πηγαίος Κώδικας:**

```
public int sw(int i){
    int j=5;
    switch(i){
        case 0: j=0;
                break;
        case 1: j=1;
                break;
        case 2: j=2;
                break;
        default: j=-1;
    }
    return j;
}
```

Συντακτικό Δέντρο:

Στην δομή switch, αν ισχύει μια περίπτωση, εκτελείται ο κώδικας που της αντιστοιχεί και με την εντολή break η ροή του προγράμματος συνεχίζει εκτός της δομής επιλογής. Αν όμως μια περίπτωση δεν ισχύει, θα πρέπει να εξεταστεί αν ισχύει η αμέσως επόμενη (εάν υπάρχει). Όπως βλέπουμε στο παρακάτω συντακτικό δέντρο, κάθε κόμβος που περιέχει μια περίπτωση, έχει δύο παιδιά. Το δεξί παιδί (κόκκινο βέλος) είναι ο κόμβος με την εντολή που θα εκτελεστεί αν ισχύει η συγκεκριμένη περίπτωση. Το αριστερό παιδί (πράσινο βέλος) είναι ο κόμβος που περιέχει την επόμενη περίπτωση (η οποία θα πρέπει να εξεταστεί αν δεν ισχύει η τρέχουσα περίπτωση). Εξάιρεση αποτελεί η περίπτωση default η οποία έχει μόνο ένα παιδί το οποίο είναι ο κόμβος με την εντολή εκτέλεσης. Αυτό συμβαίνει διότι μετά την περίπτωση default δεν ακολουθεί άλλη περίπτωση. Οι κόμβοι που περιέχουν την εντολή 'break', καθώς και ο κόμβος ο οποίος περιέχει την εντολή που εκτελείται στην περίπτωση default, έχουν μόνο ένα παιδί το οποίο είναι ο κόμβος που περιέχει το σύμβολο τερματισμού της δομής επιλογής (}).



Κεφάλαιο Δ΄

ΤΕΚΜΗΡΙΩΣΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ ΓΙΑ ΤΟΝ ΠΡΟΓΡΑΜΜΑΤΙΣΤΗ

Δ΄.1 Οι κλάσεις που περιέχει ο πηγαίος κώδικας της ε- φαρμογής

Η εφαρμογή αποτελείται από δύο πακέτα: 1) το πακέτο javassist (το οποίο περιέχει τον πηγαίο κώδικα της ήδη υπάρχουσας βιβλιοθήκης javassist η οποία χρησιμοποιείται κατά κόρον στην εφαρμογή και 2) το πακέτο jbnvd το οποίο αποτελεί τον πυρήνα της εφαρμογής.

Το πακέτο jbnvd αποτελείται από τις εξής κλάσεις:

- Main
- actionToPerform
- SourceCodeFileFilter
- AbstractSyntaxTree
- AbstractSyntaxTreeNode
- SymbolTable
- ReservedWordsPrinter
- BytecodeConversion

- Java Conversion

Δ'.2 Οι κλάσεις στην λεπτομέρεια τους

Δ'.2.1 Η κλάση Main

Η κλάση αυτή βασικά δημιουργεί το γραφικό περιβάλλον της εφαρμογής και είναι η αφετηρία για την εκτέλεση της. Αποτελείται από τις εξής μεθόδους:

- Τη μέθοδο `main(String[] args)` η οποία δημιουργεί το γραφικό περιβάλλον.
- Τη μέθοδο `getFrame()` η οποία επιστρέφει το αντικείμενο `JFrame` το οποίο αποτελεί το παράθυρο της εφαρμογής.
- Τη μέθοδο `getTextArea()` η οποία επιστρέφει το αντικείμενο `JTextArea` το οποίο περιέχει τον πηγαίο κώδικα της κλάσης που έχει επιλέξει ο χρήστης.
- Τη μέθοδο `getBytecodeTextArea()` η οποία επιστρέφει το αντικείμενο `JTextArea` το οποίο περιέχει τα bytecodes της κλάσης που έχει επιλέξει ο χρήστης.
- Τη μέθοδο `getAction()` η οποία επιστρέφει ένα αντικείμενο `actionToPerform` το οποίο ουσιαστικά καθορίζει τι θα γίνει τη στιγμή που ο χρήστης θα πατήσει το κουμπί `File Selection`.

Δ'.2.2 Η κλάση `SourceCodeFileFilter`

Η κλάση αυτή επεκτείνει την κλάση `javax.swing.filechooser.FileFilter` και φιλτράρει τα αρχεία που εμφανίζονται στον χρήστη μέσω του παραθύρου επιλογής της κλάσης, ώστε να μπορεί να διαλέξει μόνο `.java` και `.class` αρχεία.

Η κλάση αυτή έχει το πεδίο `okFileExtensions` το οποίο περιέχει τις αποδεκτές καταλήξεις αρχείων και τις εξής μεθόδους:

- Τη μέθοδο `accept(File file)` η οποία ελέγχει αν ένα αρχείο είναι τύπου `.java` ή `.class` ώστε να εμφανιστεί στο παράθυρο επιλογής αρχείων.
- Τη μέθοδο `getDescription()` η οποία δίνει μια περιγραφή των αποδεκτών αρχείων (στην περίπτωση μας η περιγραφή είναι `'Java Source Code Files'`).

Δ'.2.3 Η κλάση actionPerformed

Η κλάση αυτή υλοποιεί τη διεπαφή ActionListener και καθορίζει τις πράξεις που θα γίνουν αφότου ο χρήστης πατήσει το κουμπί File Selection. Αρχικά, η μέθοδος actionPerformed αναλαμβάνει να εμφανίσει στον χρήστη το παράθυρο επιλογής του αρχείου. Αν ο χρήστης επιλέξει ένα αρχείο .java, εμφανίζει τον πηγαίο κώδικα της κλάσης στο αριστερό παράθυρο της εφαρμογής και τέλος καλεί τη συνάρτηση convertToBytecode της κλάσης BytecodeConversion η οποία κάνει την μετατροπή σε Bytecode. Αν πάλι ο χρήστης επιλέξει ένα αρχείο .class, καλείται η συνάρτηση convertToJava της κλάσης JavaConversion η οποία αναλαμβάνει να μετατρέψει τα bytecodes της κλάσης σε πηγαίο κώδικα. Η κλάση έχει και τις υπόλοιπες μεθόδους:

- `getFilePath()` η οποία επιστρέφει ένα String το οποίο περιέχει το μονοπάτι του επιλεγμένου αρχείου.
- `getParentFilePath()` η οποία επιστρέφει ένα String το οποίο περιέχει το μονοπάτι του «γονέα» του επιλεγμένου αρχείου.
- `getFileName()` η οποία επιστρέφει ένα String το οποίο περιέχει το όνομα του επιλεγμένου αρχείου δίχως την κατάληξη του.
- `getPackageName()` η οποία επιστρέφει ένα String το οποίο περιέχει το όνομα του πακέτου στο οποίο περιλαμβάνεται η κλάση.
- `print(String line, Document doc1)` η οποία αναλαμβάνει σε περίπτωση που ο χρήστης έχει επιλέξει ένα αρχείο .java, να εμφανίσει τον πηγαίο κώδικα στο αριστερό τμήμα του παραθύρου της εφαρμογής.

Δ'.2.4 Η κλάση BytecodeConversion

Η κλάση αυτή είναι υπεύθυνη για την εμφάνιση των bytecodes της κλάσης που περιέχει το επιλεγμένο από τον χρήστη αρχείο (είτε αυτό είναι .class είτε αυτό είναι .java).

Η κλάση αποτελείται από τις εξής μεθόδους:

- `compileSourceCode(Document doc)` η οποία σε περίπτωση που ο χρήστης έχει επιλέξει ένα αρχείο .java, μεταγλωττίζει την επιλεγθείσα κλάση κάνοντας χρήση του compiler `javax.tools.compiler`.

- `convertToBytecode(Document doc)` η οποία κάνοντας χρήση συναρτήσεων του `javassist` «διαβάζει» το `.class` αρχείο και το εμφανίζει υπό τη μορφή `bytecodes` στο δεξί παράθυρο της εφαρμογής.
- `getNestedClassBytecodes(String name, Document doc)` η οποία αναλαμβάνει να εμφανίσει τα `bytecodes` των εμφωλευμένων κλάσεων που μπορεί να περιέχει η επιλεγμένη από τον χρήστη κλάση. Πρέπει να αποσαφηνιστεί ότι η μέθοδος αυτή δρα αναδρομικά. Δηλαδή μέσα στο σώμα της γίνεται ξανά κλήση της ώστε να καλυφθεί και η περίπτωση στην οποία έχουμε εμφωλευμένη κλάση μέσα στην εμφωλευμένη κλάση της επιλεγθείσας κλάσης.

Δ'.2.5 Η κλάση `ReservedWordsPrinter`

Η κλάση αποτελείται από τις εξής μεθόδους:

- `isKeyword(String word, char previous, char next)` η οποία ελέγχει αν μια λέξη που της δίνεται ως όρισμα, είναι δεσμευμένη ή όχι. Ο έλεγχος γίνεται με βάση μια δομή δεδομένων (`JSet`) η οποία περιέχει όλες τις δεσμευμένες λέξεις της γλώσσας `Java` και με βάση τον χαρακτήρα που προηγείται αλλά και τον χαρακτήρα που έπεται της λέξης.
- `printLine(String line, Document doc)` η οποία αναλαμβάνει να εμφανίσει στο αριστερό πλαίσιο κειμένου της εφαρμογής μια γραμμή του πηγαίου κώδικα του επιλεγμένου από τον χρήστη αρχείου. Η γραμμή προς εμφάνιση περνά ως όρισμα στην συνάρτηση. Η συνάρτηση ελέγχει αν υπάρχουν συμβολοσειρές στην τρέχουσα γραμμή και τις εμφανίζει με πορτοκαλί χρώμα. Επιπλέον, η συνάρτηση καλώντας την συνάρτηση `isKeyword(String word, char previous, char next)` ελέγχει την ύπαρξη δεσμευμένων λέξεων στην τρέχουσα γραμμή και τις εμφανίζει με μπλε χρώμα.

Δ'.2.6 Η κλάση `SymbolTable`

Η κλάση αυτή περιέχει μια δομή δεδομένων με την ονομασία `symbolTable` τύπου (`HashMap`) η οποία περιέχει όλα τα `bytecodes` και τις αντίστοιχες ενέργειες και δεδομένα που αντιστοιχούν σε αυτά. Η κάθε εγγραφή της δομής περιέχει την ονομασία του `bytecode` και έναν πίνακα που περιέχει δύο τιμές τύπου `String`. Η πρώτη τιμή περιγράφει την ενέργεια που θα γίνει, δηλαδή την συνάρτηση η οποία θα κληθεί αν ο `decompiler` συναντήσει το συγκεκριμένο `bytecode`. Η

δεύτερη τιμή αποτελεί το δεδομένο το οποίο θα χειριστεί η συνάρτηση η οποία θα κληθεί. Το δεδομένο αυτό, προσδιορίζει με μεγαλύτερη λεπτομέρεια την ενέργεια η οποία αντιστοιχεί στο συγκεκριμένο bytecode. Για παράδειγμα αν ο decompiler συναντήσει το bytecode `iconst_1`, τότε σύμφωνα με την δομή δεδομένων `symbolTable`, της οποίας η εγγραφή για το αντίστοιχο bytecode είναι ("`iconst_1`", new String[]{"push", "1"}), τότε θα πρέπει να κληθεί από την κλάση που αντιπροσωπεύει τον decompiler, η συνάρτηση που εκτελεί την ενέργεια `push` (ώθηση στην στοίβα τελεστών) και να χρησιμοποιηθεί ως δεδομένο σε αυτήν η τιμή 1. Το δεδομένο σε αυτήν την περίπτωση, δηλαδή σε αυτό το παράδειγμα ο αριθμός 1, αντιπροσωπεύει το στοιχείο το οποίο θα ωθηθεί στην στοίβα τελεστών. Αν για παράδειγμα πάλι ο decompiler συναντήσει το bytecode `fstore_2`, τότε σύμφωνα με την δομή δεδομένων `symbolTable`, της οποίας η εγγραφή για το αντίστοιχο bytecode είναι ("`fstore_2`", new String[]{"store", "float"}), τότε θα πρέπει να κληθεί από την κλάση που αντιπροσωπεύει τον decompiler, η συνάρτηση που εκτελεί την ενέργεια `store` (αποθήκευση τιμής σε μεταβλητή) και να χρησιμοποιηθεί ως δεδομένο σε αυτήν η λέξη `float`. Το δεδομένο σε αυτήν την περίπτωση, δηλαδή σε αυτό το παράδειγμα η λέξη `float`, αντιπροσωπεύει τον τύπο της μεταβλητής στην οποία θα γίνει η αποθήκευση τιμής.

Δ'.2.7 Η κλάση `JavaConversion`

Η κλάση αυτή είναι υπεύθυνη για την μετατροπή των bytecodes του επιλεγμένου από τον χρήστη `.class` αρχείου σε πηγαίο κώδικα Java.

Η κλάση αποτελείται από τις εξής μεθόδους:

- `convertToJava()` η οποία καλείται μόνο αν ο χρήστης έχει επιλέξει ένα `.class` αρχείο και εμφανίζει στο αριστερό παράθυρο της εφαρμογής τον σκελετό της κλάσης που περιέχει το `.class` αρχείο, τα πεδία της και τους σκελετούς όλων των constructors και των μεθόδων της υπό τη μορφή πηγαίου κώδικα Java, με την χρήση συναρτήσεων της βιβλιοθήκης `javassist`. Επίσης, η συνάρτηση αυτή, αναχτά τα bytecodes κάθε μεθόδου που περιέχει το `.class` αρχείο και καλεί τη συνάρτηση `disassembler` (για τα bytecodes κάθε μεθόδου χωριστά), ώστε να γίνει η μετατροπή τους σε πηγαίο κώδικα.
- `disassembler(List, JTextArea, List)` η οποία αναλαμβάνει να μετατρέψει τα bytecodes που έχει λάβει από τη συνάρτηση `convertToJava` σε πηγαίο κώδικα Java. Αυτό γίνεται, εξετάζοντας κάθε bytecode χωριστά και κάνοντας τις ανάλογες ενέργειες ανάλογα με την τιμή του bytecode. Η επεξεργασία ενός bytecode γίνεται με βάση την πραγματική

λειτουργία που αναπαριστά η εντολή αυτή σύμφωνα με το Java Virtual Machine Specification. Δηλαδή γίνεται μια προσομοίωση της πραγματικής λειτουργίας του bytecode ώστε να καταλήξουμε στον πηγαίο κώδικα.

- `pushAction()` η οποία χειρίζεται τα bytecodes τα οποία ωθούν μια τιμή στην στοίβα τελεστών.
- `popAction()` η οποία χειρίζεται τα bytecodes τα οποία απωθούν μια τιμή από την στοίβα τελεστών.
- `operationAction()` η οποία χειρίζεται τα bytecodes τα οποία εκτελούν μια πράξη μεταξύ δύο τελεστών.
- `storeAction()` η οποία χειρίζεται τα bytecodes τα οποία είναι υπεύθυνα για την αποθήκευση μιας τιμής σε μια μεταβλητή.
- `castAction()` η οποία χειρίζεται τα bytecodes τα οποία εφαρμόζουν typecasting (αλλαγή τύπου) σε έναν αριθμό.
- `newAction()` η οποία χειρίζεται το bytecode το οποίο δημιουργεί ένα νέο αντικείμενο μιας κλάσης.
- `creationAction()` η οποία χειρίζεται τα bytecodes τα οποία δημιουργούν έναν νέο πίνακα.
- `mutliCreationAction()` η οποία χειρίζεται το bytecode το οποίο δημιουργεί έναν πολυδιάστατο πίνακα.
- `swapAction()` η οποία χειρίζεται το bytecode το οποίο ανταλλάσσει τα δεδομένα που βρίσκονται στις δυο κορυφαίες θέσεις στη στοίβα τελεστών.
- `singleOperationAction()` η οποία χειρίζεται τα bytecodes τα οποία εφαρμόζουν τον τελεστή άρνησης σε έναν τελεστέο.
- `incrementAction()` η οποία χειρίζεται το bytecode το οποίο αυξάνει την τιμή μια μεταβλητής κατά μια σταθερή τιμή.
- `lengthAction()` η οποία χειρίζεται το bytecode το οποίο ωθεί το μέγεθος ενός πίνακα στη στοίβα τελεστών.
- `instanceAction()` η οποία χειρίζεται το bytecode το οποίο ελέγχει αν ένα αντικείμενο είναι ενός δοσμένου τύπου.

- `throwAction()` η οποία χειρίζεται το bytecode το οποίο εμφανίζει μια εξαίρεση ή ένα σφάλμα.
- `returnAction()` η οποία χειρίζεται τα bytecodes τα οποία επιστρέφουν μια τιμή ή την αναφορά σε ένα αντικείμενο.
- `getFieldAction()` η οποία χειρίζεται το bytecode το οποίο φορτώνει στη στοίβα τελεστών ένα από τα πεδία ενός αντικειμένου.
- `putFieldAction()` η οποία χειρίζεται το bytecode το οποίο αναθέτει μια τιμή σε ένα πεδίο ενός αντικειμένου.
- `getStaticAction()` η οποία χειρίζεται το bytecode το οποίο ωθεί στη στοίβα τελεστών την τιμή ενός στατικού πεδίου.
- `checkCastAction()` η οποία χειρίζεται το bytecode το οποίο ελέγχει αν ένα αντικείμενο είναι ενός δοσμένου τύπου.
- `invokeAction()` η οποία χειρίζεται τα bytecodes τα οποία καλούν μια απλή μέθοδο ή μια στατική μέθοδο ή μια μέθοδο μιας διεπαφής ή μια ειδική μέθοδο μιας κλάσης.
- `ldcAction()` η οποία χειρίζεται τα bytecodes τα οποία ωθούν στη στοίβα τελεστών αλφαριθμητικά δεδομένα που αντλούν από το constant pool.
- `dupAction()` η οποία χειρίζεται τα bytecodes τα οποία πραγματοποιούν αντιγραφή μιας ή δύο τιμών που βρίσκονται στην κορυφή της στοίβας τελεστών και τις τοποθετούν είτε στην κορυφή της στοίβας είτε, 2, 3 ή ακόμα και 4 θέσεις κάτω ανάλογα με το bytecode.
- `synchronizeAction()` η οποία χειρίζεται το bytecode το οποίο θέτει ένα αντικείμενο σε λειτουργία παρακολούθησης (monitor).
- `desynchronizeAction()` η οποία χειρίζεται το bytecode το οποίο σταματά τη διαδικασία παρακολούθησης ενός αντικειμένου (monitor).
- `switchAction()` η οποία χειρίζεται τα bytecodes τα οποία υλοποιούν τη δομή επιλογής switch.
- `branchAction()` η οποία χειρίζεται τα bytecodes τα οποία υλοποιούν τη δομή επιλογής if - else if - else ή τις δομές επανάληψης while και do - while.

- `setCurrentDoWhileNode(int branchValue, StringBuilder logicalOperator)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `branchAction` όταν αναγνωριστεί από τον αλγόριθμο ότι υπάρχει μια δομή `do-while`. Συγκεκριμένα, η συνάρτηση αναλαμβάνει να εντοπίσει την θέση στην οποία θα τοποθετηθεί ο κόμβος του συντακτικού δέντρου ο οποίος θα περιέχει την έκφραση “do{” η οποία σηματοδοτεί την αρχή της δομής `do-while`.
- `createLoopNode(AbstractSyntaxTreeNode currentNode, int branchValue)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `branchAction` όταν αναγνωριστεί από τον αλγόριθμο ότι υπάρχει μια δομή `do-while`. Συγκεκριμένα, η συνάρτηση αναλαμβάνει να δημιουργήσει και να τοποθετήσει στο συντακτικό δέντρο τον νέο κόμβο που αντιστοιχεί στην έναρξη της δομής `do-while`. Ο νέος αυτός κόμβος περιέχει ως δεδομένο την έκφραση “do{” η οποία σηματοδοτεί την αρχή μιας δομής `do-while`.
- `setDoWhileNodeData(AbstractSyntaxTreeNode currentNode, String data, String logicalOperator)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `branchAction` όταν αναγνωριστεί από τον αλγόριθμο ότι υπάρχει μια δομή `do-while`. Συγκεκριμένα, η συνάρτηση αποθηκεύει στον κόμβο του συντακτικού δέντρου που ανήκει στην συνθήκη τερματισμού της δομής `do-while`, τον αντίστοιχο πηγαίο κώδικα.
- `findDoWhileNode(AbstractSyntaxTreeNode temp, AbstractSyntaxTreeNode currentNode)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `createLoopNode` όταν χρειαστεί να εντοπιστεί στο συντακτικό δέντρο ένας ήδη υπάρχων κόμβος ο οποίος ύστερα από ελέγχους προκύπτει ότι αυτός δεν ανήκει τελικά σε δομή `do-while` και θα πρέπει συνεπώς να αλλαχθούν τα αποθηκευμένα δεδομένα του.
- `setCurrentIfNode(int branchValue, StringBuilder logicalOperator, StringBuilder leftBracket, StringBuilder rightBracket)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `branchAction` όταν αναγνωριστεί από τον αλγόριθμο ότι υπάρχει μια δομή υπόθεσης. Η συνάρτηση αναλαμβάνει να συνδέσει τον νέο κόμβο στο συντακτικό δέντρο. Η συνάρτηση καλείται επίσης και όταν ο αλγόριθμος εντοπίζει ότι υπάρχει μια συνθήκη η οποία πιθανόν να ανήκει σε μια ήδη υπάρχουσα δομή `if`. Συγκεκριμένα, η συνάρτηση αναλαμβάνει να εντοπίσει τον κόμβο στον οποίο θα πρέπει να αποθηκευτεί η νέα συνθήκη. Σημειώνεται ότι ο κόμβος αυτός περιέχει ήδη μια συνθήκη

if επομένως η νέα συνθήκη θα προστεθεί με την χρήση ενός λογικού τελεστή.

- `branchNodeConcatenation(int branchValue)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από τις συναρτήσεις `setCurrentIfNode` και `setCurrentDoWhileNode` και ο ρόλος της είναι να συνενώσει δύο κόμβους του συντακτικού δέντρου που περιέχουν αποθηκευμένες συνθήκες if ή do-while αντίστοιχα. Ο αλγόριθμος έχει εντοπίσει ότι αυτές οι συνθήκες ανήκουν στην ίδια εντολή υπόθεσης ή επανάληψης οπότε η συνάρτηση αναλαμβάνει τη συνένωση τους.
- `containsGotoStatement(int branchValue, int previousBranchValue)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `setCurrentIfNode`. Ο ρόλος αυτής της συνάρτησης είναι να εντοπίσει αν στις εντολές bytecode υπάρχει η εντολή `goto` προκειμένου να εντοπιστούν πιθανές δομές υπόθεσης ή περιπτώσεις στις οποίες θα πρέπει να γίνει συνένωση κόμβων που περιέχουν αποθηκευμένες συνθήκες if.
- `inverseRelationalOperator(String operator)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από τις συναρτήσεις `setIfNodeData` και `branchAction` και ο ρόλος της είναι να αντιστρέψει τους σχεσιακούς τελεστές. Αυτή η συνάρτηση χρειάζεται γιατί οι εντολές bytecode που αντιστοιχούν σε δομές υπόθεσης έχουν ως τελεστή τον αντίστροφο από αυτόν που έχουν οι εντολές if που τους αντιστοιχούν. Η συνθήκη δηλαδή σε μια εντολή υπόθεσης bytecode, δείχνει πότε δεν θα εκτελεστούν οι εντολές που βρίσκονται εντός του block υπόθεσης. Αντιθέτως, μια συνθήκη υπόθεσης στον πηγαίο κώδικα, δείχνει πότε θα εκτελεστούν οι εντολές που βρίσκονται μέσα στο block της υπόθεσης. Για αυτόν τον λόγο χρειάζεται και η αντιστροφή των σχεσιακών τελεστών.
- `setIfNodeData(AbstractSyntaxTreeNode currentNode, String data, String logicalOperator)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από τις συναρτήσεις `branchNodeConcatenation` και `branchAction`. Ο ρόλος αυτής της συνάρτησης είναι να αποθηκεύσει τα δεδομένα στον κόμβο του συντακτικού δέντρου που αντιπροσωπεύει τη συνθήκη της δομής υπόθεσης.
- `calculatePC(String bytecode, String currentInstruction, int currentPC)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από τη συνάρτηση `printMethod` καθώς και από την συνάρτηση `convertToBytecode(Document doc)` της κλάσης `Bytecode Conversion`. Αυτή η συνάρτηση επιστρέφει για κάθε εντολή bytecode μια ακεραία τιμή η οποία αντιστοιχεί στον αποθηκευτικό χώρο που καταλαμβάνει (με μονάδα μέτρησης το

byte). Με αυτόν τον τρόπο, η μέθοδος `convertToBytecode` είναι σε θέση να υπολογίσει τον `program counter` που αντιστοιχεί σε κάθε εντολή `bytecode` και να τον εμφανίσει αριστερά της.

- `printMethod(CtClass cc, ClassPool cp, MethodInfo minfo, Document doc, Boolean isConstructor, String className)` η οποία είναι υπεύθυνη για την εμφάνιση του πηγαιού κώδικα μιας συνάρτησης της επιλεγμένης από τον χρήστη κλάσης στο γραφικό περιβάλλον της εφαρμογής.
- `escapeCharacters(String value)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από τη συνάρτηση `ldcAction`. Όταν καλείται η συνάρτηση `ldcAction`, ωθείται μια αλφαριθμητική τιμή στην στοίβα τελεστών. Αυτή η αλφαριθμητική τιμή μπορεί να περιέχει χαρακτήρες διαφυγής (`escape characters`) και την περίπτωση αυτή έρχεται να καλύψει η συνάρτηση `escapeCharacters` καθώς οι χαρακτήρες διαφυγής απαιτούν ειδικό χειρισμό.
- `genericsHandling(int numOfArgs)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `invokeAction`. Αυτή η συνάρτηση χειρίζεται τους παραμερικούς τύπους δεδομένων (`generics`).

Δ'.2.8 Η κλάση `AbstractSyntaxTreeNode`

Αυτή η κλάση αντιπροσωπεύει τον κόμβο του συντακτικού δέντρου. Οι μέθοδοι από τις οποίες αποτελείται είναι οι `constructors` και οι μέθοδοι `set` και `get` των πεδίων.

Δ'.2.9 Η κλάση `AbstractSyntaxTree`

Αυτή η κλάση αντιπροσωπεύει το συντακτικό δέντρο. Εκτός από τον `constructor` και τις μεθόδους `set` και `get` η κλάση περιέχει και τις εξής μεθόδους:

- `addNode(Stack <AbstractSyntaxTreeNode[]> branchNodeStack, AbstractSyntaxTreeNode currentNode, AbstractSyntaxTreeNode newNode, int currentPC)` η οποία προσθέτει έναν νέο κόμβο στο συντακτικό δέντρο.
- `findNode(AbstractSyntaxTreeNode root, int pcValue)` η οποία εντοπίζει στο συντακτικό δέντρο έναν κόμβο με μια συγκεκριμένη τιμή `program counter`.

- `printTree(Document doc)` η οποία εμφανίζει τα περιεχόμενα των κόμβων του συντακτικού δέντρου.
- `popNode(Stack nodeStack)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `printTree` και απωθεί από μια στοίβα που περιέχει αποθηκευμένους κόμβους του δέντρου, τον κόμβο που βρίσκεται στην κορυφή της.
- `correctBranchNodeData(AbstractSyntaxTreeNode currentNode)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `printTree` και πραγματοποιεί διορθώσεις στον πηγαίο κώδικα προς εμφάνιση και συγκεκριμένα προσθέτει τυχόν παρενθέσεις που λείπουν.
- `optimizeBranchNodeData(AbstractSyntaxTreeNode currentNode)` η οποία είναι μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `printTree` και έχει ως στόχο να βελτιώσει τον πηγαίο κώδικα προς εμφάνιση και συγκεκριμένα να διαγράψει περιττές παρενθέσεις.

Κεφάλαιο Ε΄

Ο ΑΛΓΟΡΙΘΜΟΣ



το κεφάλαιο αυτό θα αναλύσουμε τα κύρια σημεία του αλγορίθμου δημιουργίας του λογισμικού JBVD.

Ε΄.1 Η βιβλιοθήκη Javassist

Η βιβλιοθήκη Javassist είναι ένα κουτί εργαλείων το οποίο βασίζεται στην ανακλαστική ικανότητα της Java και βοηθά στην ανάπτυξη μεταφραστών bytecodes. Είναι μια βιβλιοθήκη της Java για την τροποποίηση αρχείων class κατά τη διάρκεια της μεταγλώττισης τους ή της φόρτωσής τους. Αντίθετα με άλλες βιβλιοθήκες που δεν βασίζονται στην ανάκλαση, επιτρέπει στους χρήστες να περιγράψουν την τροποποίηση που θέλουν να πετύχουν με εντολές υψηλού επιπέδου. Αυτό σημαίνει ότι οι χρήστες δεν χρειάζεται να έχουν λεπτομερή γνώση των bytecodes ή της εσωτερικής δομής των αρχείων class της Java.

Οι χρήστες της βιβλιοθήκης Javassist μπορούν να μετατρέψουν ένα αρχείο class σε διάφορων τύπων αντικείμενα τα οποία αναπαριστούν μια κλάση, ένα πεδίο ή μια μέθοδο. Αυτά τα μεταδεδομένα είναι στη συνέχεια προσβάσιμα για μετασχηματισμό και επεξεργασία. Η εισαγωγή για παράδειγμα μιας υπερδιεπαφής ή ενός νέου πεδίου κ.ο.κ. σε μια κλάση, πραγματοποιείται μέσω της τροποποίησης αυτών των αντικειμένων. Οι αλλαγές που γίνονται σε αυτά τα μετααντικείμενα μεταφράζονται τελικά σε τροποποιήσεις του αρχείου class έτσι ώστε να επέλθει και η τροποποίηση του ορισμού της κλάσης. Από τη στιγμή που η βιβλιοθήκη Javassist δεν εκθέτει ούτε εμφανίζει τις εσωτερικές δομές δεδομένων που περιέχονται σε ένα αρχείο class, όπως για παράδειγμα ένα στοιχείο του constant pool ή η δομή των πληροφοριών μιας μεθόδου,

οι χρήστες μπορούν να την χρησιμοποιήσουν δίχως να έχουν γνώση που να αφορά τα αρχεία class της Java ή τα bytecodes.

Η βιβλιοθήκη Javassist επιτρέπει στους χρήστες να τροποποιήσουν το σώμα μιας μεθόδου. Για να αποφευχθούν προβλήματα απόδοσης, η Javassist ενημερώνει τους χρήστες όσον αφορά το πότε θα πραγματοποιηθούν λειτουργίες ανακλαστικές.

Δίνοντας ένα παράδειγμα λειτουργίας της βιβλιοθήκης, ας αναφέρουμε το αντικείμενο CtClass το οποίο παρέχεται από τη Javassist προκειμένου να αναπαραστήσει μια κλάση ενός αρχείου class. Προσφέρει σχεδόν την ίδια λειτουργικότητα με την κλάση java.lang.Class του βασικού API ανάκλασης. Για παράδειγμα, η μέθοδος getName η οποία είναι δηλωμένη στην κλάση CtClass επιστρέφει το όνομα της κλάσης, η μέθοδος getSuperclass επιστρέφει το αντικείμενο CtClass το οποίο αναπαριστά την υπερκλάση. Οι μέθοδοι getFields, getMethods, και getConstructors επιστρέφουν αντικείμενα CtField, CtMethod και CtConstructor τα οποία αναπαριστούν πεδία, μεθόδους και constructors αντίστοιχα. Τα αντικείμενα αυτά είναι ανάλογα με τα αντικείμενα java.lang.reflect.Field, Method, και Constructor. Παρέχουν διάφορες μεθόδους, όπως οι getName και getType που χρησιμεύουν στην επιθεώρηση ορισμών διαφόρων μελών μιας κλάσης. Αντίθετα με το βασικό API, η Javassist επιτρέπει στους προγραμματιστές την αλλαγή των ορισμών των κλάσεων διαμέσου αντικειμένων CtClass και συναφών κλάσεων. Για παράδειγμα, η μέθοδος setSuperclass της κλάσης CtClass αλλάζει την υπερκλάση μιας κλάσης. Η μέθοδος addMethod προσθέτει μια νέα μέθοδο στην κλάση. Ο ορισμός της νέας μεθόδου δίνεται σε μορφή αντικειμένου String που αναπαριστά το κείμενο του πηγαίου κώδικα. Η Javassist μεταγλωττίζει τον πηγαίο κώδικα σε bytecode επί τόπου και τον προσθέτει στο αρχείο class. Η μέθοδος addField προσθέτει ένα νέο πεδίο. Δέχεται κείμενο πηγαίου κώδικα που αναπαριστά την αρχική τιμή του πεδίου. Η Javassist μεταγλωττίζει τον πηγαίο κώδικα και τον εισάγει στο σώμα του constructor έτσι ώστε να αρχικοποιηθεί το πεδίο. Η μέθοδος setName της κλάσης CtClass αλλάζει το όνομα της κλάσης. Προκειμένου να υπάρξει συνέπεια, μερικές μέθοδοι όπως η setName πραγματοποιούν περισσότερες ενέργειες από το να αλλάζουν απλά ένα πεδίο ή ένα χαρακτηριστικό σε ένα αρχείο class. Για παράδειγμα, η μέθοδος setName αντικαθιστά το νέο όνομα της κλάσης οπουδήποτε εμφανίζεται στον ορισμό της κλάσης. Οι εμφανίσεις του παλιού ονόματος της κλάσης σε υπογραφές μεθόδων αλλάζουν επίσης. [10]

Ε'.2 Το γραφικό περιβάλλον

Η κλάση η οποία είναι υπεύθυνη για τη δημιουργία του γραφικού περιβάλλοντος είναι η `Main`. Προκειμένου να διαμορφωθεί το γραφικό περιβάλλον, αρχικά δημιουργείται ένα πλαίσιο (`JFrame`) του οποίου οι διαστάσεις δεν μπορούν να μεταβληθούν από τον χρήστη της εφαρμογής. Το πλαίσιο αυτό ουσιαστικά είναι το παράθυρο της εφαρμογής, το οποίο δέχεται την είσοδο και τα αποτελέσματα μετά την επεξεργασία. Ο τίτλος που δίνεται στο πλαίσιο αυτό είναι “JBVD - Java Bytecode Viewer & Decompiler - Developed By Nicole Tryfona”. Στο πλαίσιο αυτό, αντιστοιχίζεται μια δομή που καθορίζει τη διαμόρφωση του. Η δομή αυτή ανατίθεται στο πλαίσιο μέσω της δημιουργίας ενός νέου στιγμιότυπου της κλάσης `GridBagLayout`. Η κλάση `GridBagLayout` είναι βασικά ένας διαχειριστής γραφικού περιβάλλοντος ο οποίος τοποθετεί τα αντικείμενα από τα οποία απαρτίζεται το γραφικό περιβάλλον σε ένα πλέγμα γραμμών και στηλών. Στη συνέχεια, προσθέτουμε στο στιγμιότυπο `GridBagLayout` που δημιουργείται, δύο αποδέκτες περιεχομένου (`containers`) οι οποίοι είναι στιγμιότυπα της κλάσης `JPanel`. Κάθε ένα από αυτά τα `JPanel` έχουν διαστάσεις 500,500 και περιέχουν από μια περιοχή κειμένου που είναι στιγμιότυπο της κλάσης `JTextArea`. Το ένα `JPanel` έχει την ονομασία ‘Class Source Code’ και περιέχει τον πηγαίο κώδικα της κλάσης που αναλύεται. Το άλλο `JPanel` έχει τον τίτλο ‘Class Bytecode’ και περιέχει τα `bytecodes` της κλάσης που αναλύεται. Κάθε μια από τις περιοχές κειμένου δεν επιτρέπει καμία αλλαγή στο περιέχον σε αυτήν κείμενο (`is not editable`) και περιέχει και μπάρες κύλισης οι οποίες εμφανίζονται αμέσως όταν το περιέχον κείμενο ξεπερνά τις διαστάσεις της περιοχής κειμένου στην οποία ανήκει. Οι μπάρες κύλισης δημιουργούνται μέσω στιγμιότυπων της κλάσης `JScrollPane`.

Κάτω από τις περιοχές κειμένου, προστίθεται και ένα κουμπί μέσω του οποίου ο χρήστης έχει τη δυνατότητα να επιλέξει το αρχείο προς ανάλυση. Το κουμπί αυτό δημιουργείται μέσω στιγμιότυπου της κλάσης `JButton` και έχει τίτλο ‘File Selection’. Μόλις ο χρήστης πατήσει αυτό το κουμπί ενεργοποιείται ένας `ActionListener` ο οποίος προτέθηκε στο κουμπί ώστε η εφαρμογή να αντιλαμβάνεται τότε ο χρήστης πατά το κουμπί. Ο `ActionListener` στη συνέχεια μεταφέρει τον έλεγχο στην κλάση `actionToPerform`.

Η κλάση `actionToPerform` καθορίζει την ενέργεια που θα γίνει ανάλογα με το αρχείο που θα επιλέξει ο χρήστης. Συγκεκριμένα, η κλάση αυτή είναι υπεύθυνη για την εμφάνιση ενός παραθύρου διαλόγου μέσω του οποίου ο χρήστης αλληλεπιδρά και επιλέγει το αρχείο του. Αυτό επιτυγχάνεται μέσω της δημιουργίας ενός στιγμιότυπου της κλάσης `JFileChooser`. Τα αρχεία που είναι διαθέσιμα προς επιλογή από τον χρήστη φιλτράρονται από την κλάση `SourceCode-`

FileFilter η οποία περιέχει τη μέθοδο accept που ελέγχει ένα προς ένα τα αρχεία του χρήστη και τα εμφανίζει στο παράθυρο επιλογής αν και μόνο αν έχουν κατάληξη .java ή .class ή είναι φάκελοι. Έπειτα, αν ο χρήστης επιλέξει ένα αρχείο με κατάληξη .java τότε ο πηγαίος κώδικας εμφανίζεται αυτούσιος όπως ακριβώς είναι στο αρχείο που επέλεξε ο χρήστης, στην περιοχή κειμένου Class Source Code και τον έλεγχο αναλαμβάνει η κλάση BytecodeConversion η οποία θα κάνει τους απαραίτητους υπολογισμούς ώστε να εμφανιστούν τα bytecodes της κλάσης στην περιοχή κειμένου Class Bytecode. Αν από την άλλη πλευρά επιλεγεί ένα αρχείο με κατάληξη .class, η κλάση JavaConversion αναλαμβάνει να κάνει το decompilation του αρχείου και να εμφανίσει τον πηγαίο κώδικα στην περιοχή κειμένου Class Source Code ενώ η κλάση BytecodeConversion αναλαμβάνει να εμφανίσει τα bytecodes που περιέχονται στο class αρχείο στην περιοχή κειμένου Class Bytecode. [11]

Ε'.3 Η λειτουργία του bytecode viewer

Η λειτουργία του bytecode viewer ορίζεται στην κλάση BytecodeConversion. Ο bytecode viewer δέχεται ως είσοδο είτε αρχεία class τα οποία είναι έτοιμα προς ανάλυση είτε αρχεία java τα οποία θα πρέπει πρώτα να μεταγλωττιστούν προκειμένου να αναλυθούν. Αν ο χρήστης λοιπόν επιλέξει ένα αρχείο java τότε η εφαρμογή στην συνάρτηση compileSourceCode(Document doc) αναλαμβάνει να το μεταγλωττίσει και να παράγει το αρχείο class που του αντιστοιχεί χρησιμοποιώντας τον βασικό μεταγλωττιστή που προσφέρει η γλώσσα java. Ο μεταγλωττιστής αυτός καλείται μέσω της συνάρτησης getSystemJavaCompiler() της κλάσης ToolProvider η οποία επιστρέφει ένα αντικείμενο JavaCompiler. Ο bytecode viewer μεταγλωττίζει επίσης και όλες τις κλάσεις από τις οποίες εξαρτάται η κλάση που έχει επιλέξει ο χρήστης προς ανάλυση. Αν δηλαδή στην κλάση που έχει επιλέξει ο χρήστης είναι δηλωμένο ένα αντικείμενο μιας άλλης κλάσης που δεν ανήκει σε καμία από τις βιβλιοθήκες της java τότε επιβάλλεται να μεταγλωττιστεί και αυτή η κλάση. Ο τρόπος με τον οποίο ο bytecode viewer αντιλαμβάνεται τις εξαρτήσεις των κλάσεων είναι αρκετά απλός: μεταγλωττίζει επαναληπτικά την κλάση που επέλεξε ο χρήστης μέχρι να μην εμφανίζεται πλέον σφάλμα που να σχετίζεται με την έλλειψη ορισμού κλάσης. Τα σφάλματα που εμφανίζονται κατά τη μεταγλώττιση συλλέγονται από την εφαρμογή μέσω της δήλωσης ενός DiagnosticCollector και την κλήση της μεθόδου του getDiagnostics(). Όταν ο μεταγλωττιστής εντοπίζει ότι λείπει ο ορισμός μιας κλάσης επιχειρεί να βρει το αρχείο ορισμού της στο πακέτο στο οποίο ανήκει η επιλεγμένη από τον χρήστη κλάση. Αν η κλάση βρεθεί τότε η διαδικασία συνεχίζεται ενώ αν δεν βρεθεί

αυτό σημαίνει ότι έχει γίνει λάθος από το χρήστη και η μεταγλώττιση είναι αδύνατη. Επίσης, αν εντοπιστούν άλλα συντακτικά λάθη, η διαδικασία μεταγλώττισης αποτυγχάνει όπως είναι φυσικό.

Αφού ολοκληρωθεί η διαδικασία της μεταγλώττισης (σε περίπτωση που ο χρήστης επιλέξει ένα αρχείο java), ο έλεγχος της εφαρμογής μεταφέρεται στην συνάρτηση `convertToBytecode(Document doc)`. Αυτή η συνάρτηση αναλαμβάνει να εμφανίσει τις πληροφορίες και τα `bytecodes` της κλάσης του αρχείου που έχει επιλέξει ο χρήστης. Πριν ξεκινήσει η διαδικασία της ανάγνωσης του αρχείου `.class`, η εφαρμογή δημιουργεί ένα στιγμιότυπο της κλάσης `ClassPool` της βιβλιοθήκης `javassist`. Η κλάση αυτή αναπαριστά το σύνολο των κλάσεων που χρησιμοποιούνται κατά την ανάλυση ενός `class` αρχείου. Το σύνολο αυτό προσδιορίζεται μέσω του μονοπατιού του αρχείου που έδωσε ως είσοδο στην εφαρμογή ο χρήστης. Εν συνεχεία, δημιουργείται ένα στιγμιότυπο της κλάσης `CtClass` το οποίο αναπαριστά την κλάση που είναι δηλωμένη στο `class` αρχείο. Η πρόσβαση στις πληροφορίες του `class` αρχείου αποκτάται αμέσως μετά την κλήση της μεθόδου `getClassFile()` της κλάσης `CtClass`. Η κλήση της μεθόδου αυτής επιστρέφει ένα στιγμιότυπο της κλάσης `ClassFile`. Η κλάση αυτή παρέχει μεθόδους για την προσπέλαση οποιασδήποτε πληροφορίας που αφορά τα πεδία, τις μεθόδους, τους `constructors` κ.ο.κ. που είναι δηλωμένα στο `class` αρχείο. Οι πρώτες πληροφορίες τις οποίες εμφανίζει η εφαρμογή είναι οι `Major Version`, `Minor Version`, `Superclass` και `Modifiers`. Οι πληροφορίες αυτές αποκτώνται εύκολα καλώντας τις συναρτήσεις `getMajorVersion()`, `getMinorVersion()`, `getSuperclass()` και `getAccessFlags()` της κλάσης `ClassFile` αντίστοιχα.

Ύστερα, η συνάρτηση `getFields()` της κλάσης `ClassFile` δίνει όλα τα πεδία που είναι δηλωμένα στο αρχείο `class` και η εφαρμογή τα αποθηκεύει σε μια λίστα. Τα πεδία της λίστας μετατρέπονται αμέσως μετά σε αντικείμενα `FieldInfo` με `typecasting`, τα οποία καλούν τις συναρτήσεις `getName()`, `getDescriptor()` οι οποίες επιστρέφουν το όνομα και τον `descriptor` του κάθε πεδίου. Οι πληροφορίες πρόσβασης ενός πεδίου, δίνονται από τη συνάρτηση `getAccessFlags()` της κλάσης `FieldInfo`.

Αφού εμφανιστούν τα πεδία, η εφαρμογή αναλαμβάνει να εμφανίσει τις διεπαφές τις οποίες υλοποιεί η κλάση που είναι δηλωμένη στο `class` αρχείο. Αυτό επιτυγχάνεται με την κλήση της συνάρτησης `getInterfaces()` της κλάσης `ClassFile`.

Στη συνέχεια, η μέθοδος `getMethods()` της κλάσης `ClassFile` επιστρέφει όλες τις μεθόδους του αρχείου `class`. Η εφαρμογή τις αποθηκεύει σε μια λίστα. Κάθε πεδίο της λίστας μετετρέπεται με `typecasting` σε ένα αντικείμενο `MethodInfo` το οποίο περιέχει πληροφορίες για

την μέθοδο που αναπαριστά. Οι συναρτήσεις `getName()` και `getDescriptor()` επιστρέφουν το όνομα και τον descriptor της κάθε μεθόδου. Για να επιτευχθεί όμως η πρόσβαση σε άλλες βαθύτερες πληροφορίες που αφορούν τις μεθόδους επιβάλλεται να δημιουργηθεί στιγμιότυπο της κλάσης `CodeAttribute`. Το στιγμιότυπο αυτό δημιουργείται καλώντας τη συνάρτηση `getCodeAttribute()` της κλάσης `MethodInfo`. Αυτή η διαδικασία γίνεται για κάθε μέθοδο για την οποία θέλουμε να αναλύσουμε βαθύτερα. Η κλάση λοιπόν `CodeAttribute` μέσω των μεθόδων της `getMaxStack()`, `getMaxLocals()` και `getExceptionTable().size()` μας δίνει τις πληροφορίες `max stack`, `max locals` και `catch blocks` αν η μέθοδος βέβαια δεν είναι `abstract`. Οι πληροφορίες πρόσβασης της μεθόδου, δίνονται από τη συνάρτηση `getAccessFlags()` της κλάσης `MethodInfo`. Ο επόμενος στόχος της εφαρμογής αφού αποκτήσει πρόσβαση σε αυτές τις πληροφορίες είναι να εμφανίσει τα `bytecodes` της κάθε μεθόδου. Για να γίνει αυτό θα πρέπει να αποκτηθεί πρόσβαση στη δομή `constant pool` που αποθηκεύει τιμές και σταθερές που χρησιμοποιούνται από τις μεθόδους. Η πρόσβαση αποκτάται πολύ εύκολα με την κλήση της μεθόδου `getConstPool()` της κλάσης `MethodInfo`, η οποία επιστρέφει ένα στιγμιότυπο της κλάσης `ConstPool`. Εκτός αυτού, η εφαρμογή πρέπει να δημιουργήσει και ένα στιγμιότυπο της κλάσης `CodeIterator`, κάτι το οποίο επιτυγχάνει με την κλήση της συνάρτησης `iterator()` της κλάσης `CodeAttribute`. Το αντικείμενο της κλάσης `CodeIterator` διατρέχει μια προς μια τις εντολές `bytecode` της μεθόδου. Έτσι κάθε εντολή τυπώνεται στο παράθυρο της εφαρμογής και συγκεκριμένα στην περιοχή `Class Bytecode`, με τη βοήθεια της μεθόδου `instructionString(CodeIteratorInstance, current index of CodeIterator instance, constant pool)` της στατικής κλάσης `InstructionPrinter`.

Με παρόμοιο τρόπο εμφανίζονται και τα `bytecodes` των `constructors`. Συγκεκριμένα, πρώτη ενέργεια είναι η κλήση της μεθόδου `getDeclaredConstructors()` της κλάσης `CtClass`. Η μέθοδος αυτή επιστρέφει έναν πίνακα αντικειμένων `CtConstructor`, τα οποία αναπαριστούν το κάθε ένα από έναν `constructor`. Για να επιτευχθεί η πρόσβαση στις πληροφορίες ενός `constructor`, το επόμενο βήμα είναι κάθε αντικείμενο `CtConstructor` να καλέσει τη μέθοδο `getMethodInfo()`, η οποία όπως και στην περίπτωση των απλών μεθόδων επιστρέφει αντικείμενα `MethodInfo` τα οποία περιέχουν όλες τις απαραίτητες πληροφορίες. Η εμφάνιση των `bytecodes` γίνεται με τον ίδιο τρόπο που γίνεται και στις απλές μεθόδους. Δηλαδή με άλλα λόγια, αρχικά γίνεται κλήση της μεθόδου `getConstPool()` της κλάσης `MethodInfo`, η οποία επιστρέφει ένα στιγμιότυπο της κλάσης `ConstPool` και δημιουργείται ύστερα και ένα στιγμιότυπο της κλάσης `CodeIterator` μέσω κλήσης της συνάρτησης `iterator()` της κλάσης `CodeAttribute`. Το αντικείμενο της κλάσης `CodeIterator` διατρέχει μια προς μια τις εντολές `bytecode` της μεθόδου.

Έτσι κάθε εντολή τυπώνεται στο παράθυρο της εφαρμογής και συγκεκριμένα στην περιοχή Class Bytecode, με τη βοήθεια της μεθόδου `instructionString(CodeIteratorInstance, current index of CodeIterator instance, constant pool)` της στατικής κλάσης `InstructionPrinter`.

Η εφαρμογή μέσω της συνάρτησης `getNestedClassBytecodes(String name, Document doc)` της κλάσης `BytecodeConversion`, εμφανίζει ακόμα και τα bytecodes αλλά και όλες τις πληροφορίες που σχετίζονται με τις εμφωλευμένες κλάσεις. Οι εμφωλευμένες κλάσεις που είναι δηλωμένες στο class αρχείο αποκτώνται πολύ εύκολα καλώντας τη συνάρτηση `getNestedClasses()` της κλάσης `CtClass` η οποία επιστρέφει έναν πίνακα αντικειμένων `CtClass` τα οποία αναπαριστούν από μια εμφωλευμένη κλάση το καθένα. Η διαδικασία που ακολουθείται προκειμένου να εμφανιστούν οι γενικές πληροφορίες, τα πεδία, οι μέθοδοι, οι διεπαφές που υλοποιούν οι εμφωλευμένες κλάσεις και οι constructors είναι ακριβώς η ίδια διαδικασία που περιγράφηκε παραπάνω για τις απλές κλάσεις. [12]

Ε'.4 Ο decompiler

Η λειτουργία του decompiler ξεκινά από την συνάρτηση `convertToJava()`. Πριν ξεκινήσει η διαδικασία του decompiling, δημιουργείται αρχικά ένα αντικείμενο της κλάσης `ClassPool`. Όπως αναφέρθηκε και παραπάνω ένα τέτοιο αντικείμενο αναπαριστά το σύνολο των κλάσεων που χρησιμοποιούνται από την εφαρμογή. Έπειτα, δημιουργείται ένα στιγμιότυπο της κλάσης `CtClass` το οποίο συμβολίζει την κλάση προς επεξεργασία. Το αρχείο class αναπαριστάται μέσω ενός αντικειμένου της κλάσης `ClassFile` το οποίο αποκτάται καλώντας τη μέθοδο `getClassFile()` της κλάσης `CtClass`. Ξεκινώντας τη βασική λειτουργία του ο decompiler βρίσκει το όνομα της κλάσης που ορίζεται στο αρχείο class που επέλεξε ο χρήστης και το πακέτο στο οποίο ανήκει καλώντας τη συνάρτηση `getName()` της κλάσης `ClassFile`. Οι κλάσεις οι οποίες εισάγονται (`import`) στο επιλεγμένο αρχείο class αποκτώνται πολύ απλά καλώντας τη μέθοδο `getRefClasses()` της κλάσης `CtClass`. Αυτή η μέθοδος επιστρέφει όλες τις κλάσεις που εισάγονται συμπεριλαμβανομένων και των κλάσεων που ανήκουν στο πακέτο `java.lang`. Όμως από τη στιγμή που το πακέτο αυτό γίνεται `import` πάντα εξόρισμού η εφαρμογή δεν εμφανίζει τις κλάσεις αυτές στα imports. Οι πληροφορίες πρόσβασης της κλάσης δίνονται από τη συνάρτηση `getAccessFlags()` της κλάσης `ClassFile`. Αν η υπερκλάση της κλάσης που ορίζεται στο class αρχείο δεν είναι η `java.lang.Object`, η οποία είναι υπερκλάση όλων των κλάσεων, εμφανίζεται μέσω της συνάρτησης `getSuperclass()` της κλάσης `ClassFile`. Αναλόγως, οι διεπαφές τις οποίες υλοποιεί η επιλεγμένη κλάση είναι διαθέσιμες στην εφαρμογή

μέσω της μεθόδου `getInterfaces()` της κλάσης `ClassFile`. Εν συνεχεία, η εφαρμογή προχωρά στην εμφάνιση των πεδίων της κλάσης. Η συνάρτηση `getFields()` της κλάσης `ClassFile` η οποία επιστρέφει μια λίστα με όλα τα πεδία είναι κατάλληλη για αυτό. Κάθε πεδίο που υπάρχει στη λίστα μετατρέπεται με `typecasting` σε αντικείμενο `FieldInfo`. Οι πληροφορίες πρόσβασης του εμφανίζονται καλώντας τη μέθοδο `getAccessFlags()` της κλάσης `FieldInfo`. Με παρόμοιο τρόπο, δηλαδή με κλήση της συνάρτησης `getDescriptor()` της κλάσης `FieldInfo`, εμφανίζεται και ο `descriptor` του πεδίου. Σημειώνεται ότι αν ο `descriptor` περιέχει το σύμβολο '[' τότε αυτό σημαίνει ότι το πεδίο είναι πίνακας. Ομοίως αν ο `descriptor` περιέχει το 'I', το 'J', το 'D', το 'F', το 'S', το 'B', το 'Z' ή το 'C', τότε το πεδίο είναι `int`, `long`, `double`, `float`, `short`, `byte`, `boolean` ή `char` αντίστοιχα. Τέλος, το όνομα του πεδίου αποκτήνεται μέσω κλήσης της συνάρτησης `getName()` της κλάσης `FieldInfo`.

Επόμενο βήμα της εφαρμογής είναι η δημιουργία του σκελετού των `constructors` της κλάσης. Οι `constructors` επιστρέφονται σε έναν πίνακα ως αντικείμενα της κλάσης `CtConstructor` μέσω της μεθόδου `getDeclaredConstructors()` της κλάσης `CtClass`. Κάθε αντικείμενο του πίνακα αυτού μετατρέπεται με `typecasting` σε αντικείμενο `MethodInfo`. Στη συνέχεια, οι `constructors` εμφανίζονται στο δεξί πλαίσιο της εφαρμογής μέσω κλήσης της συνάρτησης `printMethod(CtClass cc, ClassPool cp, MethodInfo minfo, Document doc, Boolean isConstructor, String className)`. Οι πληροφορίες πρόσβασης ενός `constructor` αποκτήνται πολύ απλά με την κλήση της μεθόδου `getAccessFlags()` της κλάσης `MethodInfo` ενώ το όνομα του μέσω κλήσης της μεθόδου `getName()` της ίδιας κλάσης. Το επόμενο πράγμα που πρέπει να κάνει η εφαρμογή είναι να βρει τα ορίσματα που έχει ο κάθε `constructor`. Τα ορίσματα δίνονται από τη συνάρτηση `getDescriptor()` της κλάσης `MethodInfo` και περιλαμβάνονται σε ένα ζεύγος παρενθέσεων. Οπότε με ειδική επεξεργασία η εφαρμογή κρατά ως ορίσματα το περιεχόμενο των παρενθέσεων μόνο διότι αν πρόκειται για μέθοδο και όχι για `constructor` τότε δίπλα από το ζεύγος παρενθέσεων παρατίθεται και ο τύπος επιστροφής της. Εδώ δεν έχουμε όπως είναι φανερό τύπο επιστροφής αφού ασχολούμαστε με `constructors` και όχι με απλές μεθόδους. Στην περίπτωση των `constructors` η λογική για την ανίχνευση των ορισμάτων είναι ίδια με αυτήν που ακολουθείται για την ανίχνευση των τύπων των πεδίων. Δηλαδή, αν στο περιεχόμενο των παρενθέσεων περιλαμβάνεται ο χαρακτήρας 'I', ο 'J', ο 'D', ο 'F', ο 'S', ο 'B', ο 'Z' ή ο 'C', τότε το πεδίο είναι `int`, `long`, `double`, `float`, `short`, `byte`, `boolean` ή `char` αντίστοιχα. Σημειώνεται ότι αν πριν από ένα όρισμα υπάρχει ο χαρακτήρας '[', τότε αυτό σημαίνει ότι αυτό αντιπροσωπεύει έναν πίνακα όπως αναφέρθηκε και παραπάνω. Αν ένα όρισμα δεν είναι ένος βασικού τύπου της `java` αλλά είναι στιγμιότυπο μιας κλάσης όπως για παράδειγμα στην περίπτωση

(Ljava/lang/String;) όπου το πρώτο όρισμα είναι ακέραιος (φαίνεται από το I) και το δεύτερο είναι String (Ljava/lang/String) τότε η εφαρμογή το εντοπίζει μέσω των χαρακτήρων '/' που υπάρχουν στο πλήρες όνομα της κλάσης και από το ελληνικό ερωτηματικό που βρίσκεται στο τέλος του ονόματος. Είναι αξιοσημείωτο ότι ο decompiler δεν εμφανίζει το πλήρες όνομα της κλάσης στα ορίσματα αλλά το απλό. Για παράδειγμα, στην παραπάνω περίπτωση θα εμφανίσει: (int var1, String var2). Αφού δημιουργηθεί ο σκελετός ενός constructor αυτό που πρέπει να γίνει είναι να αποκτηθούν τα bytecodes που αντιστοιχούν στον εκτελέσιμο κώδικα του. Αυτό γίνεται αρχικά μέσω της δημιουργίας ενός αντικειμένου της κλάσης CodeAttribute. Το αντικείμενο αυτό δημιουργείται μέσω κλήσης της συνάρτησης getCodeAttribute() της κλάσης MethodInfo. Στη συνέχεια καλείται η συνάρτηση getConstPool() της κλάσης MethodInfo η οποία επιστρέφει ένα αντικείμενο ConstPool το οποίο συμβολίζει τη δομή constant pool ενός class αρχείου, η οποία περιέχει σταθερές και αλφαριθμητικά δεδομένα όπως Strings και άλλα. Τέλος, η μέθοδος iterator() της κλάσης CodeAttribute μας επιστρέφει ένα αντικείμενο CodeIterator το οποίο περιέχει και τις εντολές bytecode προς ανάλυση. Η ανάλυση των εντολών γίνεται μέσω κλήσης της συνάρτησης disassembler(CtClass cc, MethodInfo minfo, Document doc).

Τελειώνοντας με τους constructors, σειρά έχουν οι μέθοδοι. Η δημιουργία του σκελετού μιας μεθόδου είναι σχεδόν ίδια με αυτή που ακολουθείται για τους constructors και επιτυγχάνεται και αυτή μέσω κλήσης της συνάρτησης printMethod(CtClass cc, ClassPool cp, MethodInfo minfo, Document doc, Boolean isConstructor, String className).. Για το λόγο αυτό θα αναφερθούν μόνο οι διαφορές οι οποίες είναι και ελάχιστες. Η εφαρμογή αρχικά αποκτά πρόσβαση στις μεθόδους μέσω της μεθόδου getMethods() της κλάσης ClassFile. Επισημαίνεται ότι η μέθοδος αυτή επιστρέφει όλες τις μεθόδους συμπεριλαμβανομένων και των ειδικών όπως οι constructors. Ο decompiler παίρνει το όνομα κάθε μεθόδου μέσω της συνάρτησης getName() της κλάσης MethodInfo και αν αυτό είναι διάφορο του <init>, το οποίο σημαίνει ότι πρόκειται για constructor, τότε συνεχίζει τη διαδικασία. Σε διαφορετική περίπτωση προχωρά στην επόμενη μέθοδο. Η κύρια διαφορά με τον σκελετό των constructors έγκειται στις εξαιρέσεις. Η εφαρμογή λοιπόν βρίσκει τις εξαιρέσεις που 'πετά' μια μέθοδος καλώντας τη συνάρτηση getExceptionsAttribute() της κλάσης MethodInfo, η οποία επιστρέφει ένα αντικείμενο της κλάσης ExceptionsAttribute. Στη συνέχεια, το αντικείμενο αυτό καλεί τη μέθοδο getExceptions() η οποία επιστρέφει έναν πίνακα με όλες τις εξαιρέσεις. Η τελευταία διαφορά με τον σκελετό των constructors αφορά τον τύπο επιστροφής που όπως ειπώθηκε παραπάνω απουσιάζει από έναν constructor. Ο τύπος αυτός βρίσκεται αμέσως μετά από τα ο-

ρίσματα τα οποία περικλείονται από ένα ζεύγος παρενθέσεων. Για παράδειγμα στον descriptor (Ljava/lang/String;)I τα ορίσματα περικλείονται στην παρένθεση και αμέσως μετά ακολουθεί ένα I το οποίο δηλώνει ότι η μέθοδος επιστρέφει έναν ακέραιο αριθμό.

Στη συνέχεια ο decompiler ξεκινά την ανάλυση των bytecodes με στόχο την απόκτηση του πηγαίου κώδικα. Η λειτουργία αυτή επιτυγχάνεται με την εκτεταμένη χρήση ενός συντακτικού δέντρου (AbstractSyntaxTree). Οι κόμβοι του δέντρου δημιουργούνται δυναμικά κατά τη διάρκεια ανάλυσης των bytecodes από την εφαρμογή. Σε κάθε κόμβο του δέντρου αποθηκεύεται και μια εντολή του πηγαίου κώδικα. Το συνακτικό δέντρο αντιπροσωπεύεται από την κλάση (AbstractSyntaxTree) ενώ ένας κόμβος του αντιπροσωπεύεται από την κλάση (AbstractSyntaxTreeNode). Σε κάθε κόμβο του δέντρου αποθηκεύεται η τρέχουσα εντολή, ένας δείκτης για τον αριστερό κόμβο και ένας δείκτης για τον δεξί κόμβο. Μια ακόμη δομή η οποία κυριαρχεί στην διαδικασία ανάκτησης του πηγαίου κώδικα είναι η στοίβα τελεστών. Σε αυτή τη δομή αποθηκεύονται προσωρινά αποτελέσματα και απωθούνται τα δεδομένα που χρειάζονται για όλους σχεδόν τους υπολογισμούς. Τέλος, είναι πολύ σημαντικό να αναφερθεί ότι η κλάση SymbolTable περιέχει μια δομή δεδομένων με την ονομασία symbolTable τύπου (HashMap) η οποία περιέχει όλα τα bytecodes και τις αντίστοιχες ενέργειες και δεδομένα που αντιστοιχούν σε αυτά. Η κάθε εγγραφή της δομής περιέχει την ονομασία του bytecode και έναν πίνακα που περιέχει δύο τιμές τύπου String. Η πρώτη τιμή περιγράφει την ενέργεια που θα γίνει, δηλαδή την συνάρτηση η οποία θα κληθεί αν ο decompiler συναντήσει το συγκεκριμένο bytecode. Η δεύτερη τιμή αποτελεί το δεδομένο το οποίο θα χειριστεί η συνάρτηση η οποία θα κληθεί. Το δεδομένο αυτό, προσδιορίζει με μεγαλύτερη λεπτομέρεια την ενέργεια η οποία αντιστοιχεί στο συγκεκριμένο bytecode.

Όταν καλείται η συνάρτηση disassembler(CtClass cc, MethodInfo minfo, Document doc) η οποία αναλύει μια προς μια τις εντολές bytecode μιας μεθόδου, τότε για κάθε Bytecode γίνεται αναζήτηση στην δομή δεδομένων SymbolTable για την εγγραφή που του αντιστοιχεί και αποθηκεύεται στην μεταβλητή action η ενέργεια που θα γίνει και στην μεταβλητή symbol το δεδομένο το οποίο προσδιορίζει με μεγαλύτερη λεπτομέρεια το είδος αυτής της ενέργειας. Για κάθε ενέργεια έχει δημιουργηθεί και η αντίστοιχη συνάρτηση η οποία αναλαμβάνει να την εκτελέσει. Το όνομα αυτών των συναρτήσεων αποτελείται από την συνένωση του περιεχομένου της μεταβλητής action με την λέξη "Action". Για παράδειγμα αν η μεταβλητή action έχει την τιμή push, τότε το όνομα της συνάρτησης που θα φέρει εις πέρας την αντίστοιχη ενέργεια είναι pushAction. Με αυτόν τον τρόπο η εφαρμογή μπορεί και καλεί με ευκολία την συνάρτηση που αντιστοιχεί στην ανάλυση του κάθε bytecode. Για κάθε εντολή bytecode

που η συνάρτηση `disassembler(CtClass cc, MethodInfo minfo, Document doc)` καλείται να αναλύσει, γίνεται κλήση της αντίστοιχης συνάρτησης.

Παρακάτω αναλύονται οι συναρτήσεις που φέρνουν εις πέρας τις ενέργειες που αντιστοιχούν σε κάθε bytecode:

Συνάρτηση `pushAction`

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: `iconst_m1`, `iconst_0`, `iconst_1`, `iconst_2`, `iconst_3`, `iconst_4`, `iconst_5`, `iconst`, `lconst_0`, `lconst_1`, `lconst`, `bipush`, `sipush`, `iload`, `lload`, `fload`, `dload`, `aload`, `aload_0`, `fconst_0`, `dconst_0`, `fconst_1`, `dconst_1`, `fconst_2`, `iload_0`, `lload_0`, `fload_0`, `dload_0`, `iload_1`, `aload_1`, `lload_1`, `fload_1`, `dload_1`, `iload_2`, `aload_2`, `lload_2`, `fload_2`, `dload_2`, `iload_3`, `aload_3`, `lload_3`, `fload_3`, `dload_3`, `aconst_null`, `iaload`, `laload`, `faload`, `daload`, `caload`, `saload`, `baload`, `aaload`. Στην ενέργεια `push`, το σύμβολο `symbol` αντιπροσωπεύει την τιμή προς ώθηση στην στοίβα τελεστών. Συγκεκριμένα, αν το σύμβολο είναι η τιμή `null`, τότε στην στοίβα τελεστών ωθείται το όνομα μιας ήδη υπάρχουσας μεταβλητής. Το όνομα προκύπτει από την συνένωση του λεκτικού `var` με τον αριθμό που βρίσκεται στην εντολή bytecode αμέσως μετά τον κενό χαρακτήρα. Για παράδειγμα αν έχουμε την εντολή `aload 4`, τότε θα γίνει ώθηση της τιμής `var4` στην στοίβα τελεστών. Αν το σύμβολο είναι ο χαρακτήρας `[`, τότε η τιμή προς ώθηση είναι ένα στοιχείο ενός πίνακα. Η αναφορά στον πίνακα `arrayref` καθώς και ο δείκτης (`index`) στο στοιχείο του πίνακα, απωθούνται πρώτα από την στοίβα τελεστών. Στη συνέχεια στην στοίβα τελεστών ωθείται το όνομα της μεταβλητής που αντιπροσωπεύει το στοιχείο του πίνακα (`arrayref + "[" + index + "]"`). Στην περίπτωση που το σύμβολο είναι το `'`, τότε ωθείται στην στοίβα τελεστών ο αριθμός που βρίσκεται στην εντολή bytecode αμέσως μετά τον κενό χαρακτήρα. Για παράδειγμα αν έχουμε την εντολή `bipush 6`, τότε θα γίνει ώθηση της τιμής `6` στην στοίβα τελεστών. Τέλος, αν το σύμβολο είναι το `"this"`, τότε ωθείται στην λίστα τελεστών το όνομα της μεταβλητής `var0` αν αυτή υπάρχει ήδη ή σε διαφορετική περίπτωση ωθείται το λεκτικό `"this"`. Σε οποιαδήποτε άλλη περίπτωση ωθείται το ίδιο το σύμβολο στην στοίβα τελεστών, το οποίο είναι είτε μια αριθμητική τιμή είτε το όνομα μιας ήδη υπάρχουσας μεταβλητής.

Συνάρτηση `popAction`

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: `pop`, `pop2`. Στην ενέργεια `pop` απωθείται από την στοίβα τελεστών η τιμή που βρίσκεται στην κορυφή της. Στη συνέχεια δημιουργείται και προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο και αποθηκεύεται στο πεδίο του `data` η τιμή που έχει απωθηθεί μαζί με το σύμβολο `';` το οποίο αντιπροσωπεύει το τέλος μιας

εντολής στην Java.

Συνάρτηση operationAction

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: iadd, ladd, fadd, dadd, isub, lsub, fsub, dsub, imul, lmul, fmul, dmul, idiv, ldiv, fdiv, ddiv, irem, lrem, frem, drem, ior, lor, iand, land, ixor, lxor, ishl, lshl, ishr, lshr, iushr, lushr. Στην ενέργεια operation, το σύμβολο symbol αντιπροσωπεύει το είδος της πράξης που θα γίνει μεταξύ δύο τελεστών. Οι δύο τελεστοί op1 και op2 απωθούνται από την στοίβα τελεστών και ωθείται η εντολή της πράξης που εκτελείται μεταξύ τους: (op1 + symbol + op2).

Συνάρτηση storeAction

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: dstore_0, dstore_1, dstore_2, dstore_3, dstore, fstore_0, fstore_1, fstore_2, fstore_3, fstore, istore_0, istore_1, istore_2, istore_3, istore, lstore_0, lstore_1, lstore_2, lstore_3, lstore, astore_0, astore_1, astore_2, astore_3, astore. Στην ενέργεια store το σύμβολο αντιπροσωπεύει τον τύπο της μεταβλητής στην οποία θα γίνει η αποθήκευση της νέας τιμής. Η τιμή προς αποθήκευση value απωθείται από την στοίβα τελεστών. Αν το σύμβολο είναι το '[', τότε πρόκειται για αποθήκευση τιμής σε ένα στοιχείο ενός πίνακα. Η αναφορά στον πίνακα arrayref καθώς και ο δείκτης index στο στοιχείο του πίνακα, απωθούνται από την στοίβα τελεστών. Στη συνέχεια δημιουργείται και προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο ο οποίος έχει ως δεδομένο την εντολή αποθήκευσης του νέου στοιχείου του πίνακα: arrayref[index] = value;. Αν η αναφορά στον πίνακα περιέχει το λεκτικό new, τότε δεν ακολουθείται η παραπάνω διαδικασία αλλά ωθείται στην στοίβα τελεστών η εξής έκφραση "arrayref {value}". Σημειώνεται ότι αν η αναφορά arrayref εκτός από το λεκτικό new περιέχει και το σύμβολο '{', τότε στην στοίβα τελεστών ωθείται η έκφραση "arrayref.substring(0, arrayref.length()-1), value)".

Στην περίπτωση που το σύμβολο δεν είναι το null και δεν ισούται με '[', τότε πρόκειται για την αποθήκευση τιμής σε μια απλή μεταβλητή. Σε αυτήν την περίπτωση δημιουργείται και προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο. Ο κόμβος αυτός περιέχει σαν δεδομένο την εντολή αποθήκευσης της νέας τιμής στην μεταβλητή. Αν η μεταβλητή αυτή δεν υπάρχει (αν το index είναι μεγαλύτερο του currentVar που έχει αποθηκευμένο τον αύξοντα αριθμό της τελευταίας δηλωμένης μεταβλητής), τότε αποθηκεύεται ως δεδομένο στον κόμβο του δέντρου η εντολή δήλωσης της "symbol var + index = value;". Υπενθυμίζεται ότι το symbol αντιπροσωπεύει τον τύπο της μεταβλητής. Στο index αποθηκεύεται ο αύξων αριθμός της μεταβλητής στην οποία θα γίνει η αποθήκευση. Η τιμή αυτή υπάρχει και εκμαιεύεται από την εντολή

bytecode. Επειδή η μεταβλητή αυτή είναι καινούργια, αποθηκεύεται στον πίνακα κατακερματισμού varTypes, ο οποίος περιέχει όλους τους τύπους των μεταβλητών που έχουν δηλωθεί, ο τύπος της. Σε περίπτωση που η μεταβλητή έχει δηλωθεί και υπάρχει ήδη, τότε στον κόμβο του δέντρου αποθηκεύεται απλά η εντολή μεταβολής της τιμής της “var + index = value;”. Τέλος, σε περίπτωση που το σύμβολο ισούται με null, τότε πρόκειται για αποθήκευση τιμής σε μεταβλητή αναφοράς αντικειμένου. Ομοίως και στην περίπτωση αυτή δημιουργείται και αποθηκεύεται στο συντακτικό δέντρο ένας νέος κόμβος που περιέχει ως δεδομένο της εντολής ανάθεσης τιμής στην μεταβλητή. Αν η μεταβλητή αναφοράς αντικειμένου δεν έχει δηλωθεί ήδη, τότε πρέπει να γίνει αποθήκευση της στον πίνακα κατακερματισμού varTypes. Ο τύπος της μεταβλητής αναφοράς αντικειμένου προκύπτει ως εξής: αν στην τιμή προς αποθήκευση value υπάρχει το σύμβολο του διπλού εισαγωγικού, τότε πρόκειται για αντικείμενο τύπου String, οπότε αποθηκεύεται στον κόμβο του δέντρου η τιμή “String var + index = value;”. Αν η τιμή προς αποθήκευση ισούται με null, τότε ο τύπος του αντικειμένου είναι ο Object και αποθηκεύεται στον κόμβο του δέντρου η εντολή “Object var + index = value;”. Τέλος, αν η τιμή προς αποθήκευση περιέχει το λεκτικό new, τότε ο τύπος της μεταβλητής αναφοράς αντικειμένου υπάρχει στην εντολή bytecode και εκμαιεύεται ύστερα από επεξεργασία αυτής. Στον κόμβο του δέντρου αποθηκεύεται η εντολή “type var + index = value;”.

Συνάρτηση castAction

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: d2f, d2i, d2l, f2d, f2i, f2l, i2b, i2b, i2d, i2f, i2l, i2s, l2d, l2f, l2i. Αυτά τα bytecodes κάνουν typecasting του αριθμού που βρίσκεται στην κορυφή της στοίβας τελεστών. Στην ενέργεια cast, το σύμβολο symbol αντιπροσωπεύει το είδος της μετατροπής τύπου που πρόκειται να γίνει. Ο τελεστέος num στον οποίο θα εκτελεστεί το typecasting, απωθείται από την στοίβα τελεστών και ωθείται η εντολή της μετατροπής τύπου του: ((symbol)num). Αν για παράδειγμα το bytecode είναι το i2l, τότε πρόκειται για μετατροπή τύπου του αμέριστου αριθμού num που βρίσκεται στην κορυφή της στοίβας τελεστών, από int σε long και θα ωθηθεί στην στοίβα τελεστών η έκφραση ((long)num).

Συνάρτηση newAction

Αυτή η συνάρτηση χειρίζεται το bytecode new. Αυτό το bytecode δημιουργεί ένα νέο αντικείμενο. Η κλάση στην οποία ανήκει το νέο αντικείμενο, έστω class, εκμαιεύεται από την ίδια την εντολή. Αν για παράδειγμα η εντολή είναι “new #2 = Class client.Client”, τότε η κλάση στην οποία ανήκει το νέο αντικείμενο είναι η κλάση Client του πακέτου client. Στη συνέχεια

ωθείται στη στοίβα τελεστών η παράσταση “new class”, όπου class η ονομασία της κλάσης στην οποία ανήκει το νέο αντικείμενο.

Συνάρτηση createAction

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: newarray, anewarray. Αυτά τα bytecodes δημιουργούν έναν καινούριο πίνακα. Στην ενέργεια creation, το σύμβολο symbol αντιπροσωπεύει το είδος του πίνακα που θα δημιουργηθεί και συγκεκριμένα αν θα είναι ένας πίνακας απλών κανονικών τύπων ή ένας πίνακας αναφορών. Ο τύπος του πίνακα που θα δημιουργηθεί, έστω type, εκμαιεύεται από την ίδια την εντολή, όπως για παράδειγμα από την εντολή “newarray int” φαίνεται ότι θα δημιουργηθεί ένας πίνακας ακεραίων ή από την εντολή “anewarray #2 = Class java.lang.String” φαίνεται ότι θα δημιουργηθεί ένας πίνακας από String. Το μέγεθος του πίνακα, έστω length, απωθείται από τη στοίβα τελεστών. Τέλος, ωθείται στη στοίβα τελεστών η έκφραση “new type[length]”.

Συνάρτηση multiCreationAction

Αυτή η συνάρτηση χειρίζεται το bytecode multianewarray. Αυτό το bytecode δημιουργεί έναν πολυδιάστατο πίνακα. Οι διαστάσεις του πίνακα formedDims προκύπτουν από το πλήθος των χαρακτήρων '[' που υπάρχουν στην εντολή. Αν για παράδειγμα έχουμε την εντολή “multianewarray #2 = Class [[Ljava.lang.String;” τότε οι διαστάσεις είναι δυο αφού υπάρχουν δυο χαρακτήρες '['. Το μέγεθος της κάθε διάστασης απωθείται από τη στοίβα τελεστών. Ο τύπος του πίνακα typeFormed που θα δημιουργηθεί προκύπτει επίσης από την εντολή. Στο παράδειγμα μας ο τύπος είναι String. Αν όμως στη θέση του Ljava.lang.String υπήρχε ένας από τους χαρακτήρες I, J, D, F, S, B, ή Z τότε ο τύπος του πίνακα θα ήταν int, long, double, float, short, byte, boolean ή char αντίστοιχα. Τέλος αν υποθέσουμε ότι ο τύπος του πίνακα είναι type, οι διαστάσεις του είναι n και το μέγεθος της κάθε μιας είναι size_i, τότε ωθείται στη στοίβα τελεστών η έκφραση “new type[size₁][size₂]...[size_n]”.

Συνάρτηση swapAction

Αυτή η συνάρτηση χειρίζεται το bytecode swap. Αυτό το bytecode ανταλλάσσει τα δεδομένα που βρίσκονται στις δυο κορυφαίες θέσεις στη στοίβα τελεστών. Επομένως απωθούνται από τη στοίβα οι δυο κορυφαίες τιμές έστω op1 η πρώτη και op2 η δεύτερη και στη συνέχεια ωθείται πρώτα η op1 και στο τέλος η op2 ώστε να βρίσκεται στην κορυφή.

Συνάρτηση singleOperationAction

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: ineg, lneg, fneg, dneg. Αυτά τα bytecodes

απωθούν τον αριθμό που βρίσκεται στην κορυφή της στοίβας τελεστών με τιμή έστω op1. Στη συνέχεια ωθούν στη στοίβα τελεστών την παράσταση “(~op1)”.

Συνάρτηση incrementAction

Αυτή η συνάρτηση χειρίζεται το bytecode iinc. Αυτό το bytecode αυξάνει ή μειώνει την τιμή μιας τοπικής μεταβλητής κατά μια σταθερή τιμή. Αν για παράδειγμα έχουμε την εντολή iinc 2, 1 τότε αυτό σημαίνει ότι η τοπική μεταβλητή var2 πρέπει να αυξηθεί κατά 1. Άρα η αντίστοιχη εντολή στην Java είναι var2 += 1. Κατά τη διάρκεια της ανάλυσης του bytecode iinc δημιουργείται και προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο και αποθηκεύεται στο πεδίο του data, ανάλογα με το αν έχουμε αύξηση ή μείωση της τιμής, η εντολή “var_n += value;” ή η εντολή “var_n -= value” αντίστοιχα, όπου n είναι ο δείκτης της τοπικής μεταβλητής που θα αυξηθεί και value είναι η τιμή μεταβολής.

Συνάρτηση lenghtAction

Αυτή η συνάρτηση χειρίζεται το bytecode arraylength. Αυτό το bytecode ωθεί το μέγεθος ενός πίνακα στη στοίβα τελεστών. Η αναφορά του πίνακα, έστω arrayref, απωθείται από τη στοίβα τελεστών. Η έκφραση arrayref.length ωθείται στη συνέχεια στη στοίβα τελεστών.

Συνάρτηση instanceofAction

Αυτή η συνάρτηση χειρίζεται το bytecode instanceof. Αυτό το bytecode ελέγχει αν ένα αντικείμενο είναι ενός δοσμένου τύπου. Η αναφορά στο αντικείμενο, έστω ref, απωθείται από τη στοίβα τελεστών. Ο τύπος προκύπτει από την ίδια την εντολή και έστω ότι είναι className. Αν για παράδειγμα έχουμε την εντολή “instanceof java.lang.String”, τότε ο τύπος είναι String. Στη συνέχεια ωθείται στη στοίβα τελεστών η έκφραση “ref instanceof className”.

Συνάρτηση throwAction

Αυτή η συνάρτηση χειρίζεται το bytecode athrow. Αυτό το bytecode εμφανίζει εξαίρεση ή σφάλμα. Η αναφορά στο Throwable αντικείμενο απωθείται από τη στοίβα τελεστών και έστω ότι είναι throwValue. Κατά τη διάρκεια της ανάλυσης του bytecode athrow δημιουργείται και προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο και αποθηκεύεται στο πεδίο του data η έκφραση “throw throwValue;”.

Συνάρτηση returnAction

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: ireturn, lreturn, freturn, dreturn, areturn. Αυτά τα bytecodes επιστρέφουν μια τιμή ή την αναφορά σε ένα αντικείμενο. Η τιμή ή η

αναφορά που θα επιστραφεί απωθείται από την κορυφή της στοίβας τελεστών και έστω ότι αντιπροσωπεύεται από την μεταβλητή `obj` τύπου `Object`. Κατά τη διάρκεια της ανάλυσης αυτών των `bytecodes` δημιουργείται και προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο και αποθηκεύεται στο πεδίο του `data` η έκφραση `"return obj;"`.

Συνάρτηση `getFieldAction`

Αυτή η συνάρτηση χειρίζεται το `bytecode` `getfield`. Αυτό το `bytecode` φορτώνει στη στοίβα τελεστών ένα από τα πεδία ενός αντικειμένου. Το πεδίο εκμαιεύεται από την ίδια την εντολή. Αν για παράδειγμα η εντολή είναι `"getstatic #3 = Field Item.uniqueId(I)"`, τότε το πεδίο προς φόρτωση είναι το `uniqueId`. Η αναφορά στο αντικείμενο στο οποίο ανήκει το πεδίο απωθείται από τη στοίβα τελεστών. Αν η αναφορά έχει τιμή `ref` και το πεδίο έχει τιμή `field` τότε η έκφραση `ref.field` ωθείται στη στοίβα τελεστών.

Συνάρτηση `putFieldAction`

Αυτή η συνάρτηση χειρίζεται το `bytecode` `putfield`. Αυτό το `bytecode` αναθέτει μια τιμή σε ένα πεδίο ενός αντικειμένου απωθώντας από τη στοίβα τελεστών την αναφορά στο αντικείμενο, έστω `objectref` και την τιμή, έστω `value`. Το πεδίο προς ενημέρωση προκύπτει από την εντολή και έστω ότι η ονομασία του είναι `field`. Αν για παράδειγμα έχουμε την εντολή `"putfield #2 = Field javassist.javassistextended.Hello.s(S)"`, τότε το πεδίο προς ενημέρωση είναι το `s` της κλάσης `Hello`, του υποπακέτου `javassistextended` το οποίο ανήκει στο πακέτο `javassist`. Επίσης η εντολή μας πληροφορεί ότι το πεδίο είναι τύπου `short` (ο τύπος `S`, που μεταφράζεται ως `short`, περιλαμβάνεται σε ένα ζεύγος παρενθέσεων). Κατά τη διάρκεια της ανάλυσης του `bytecode` `putField` δημιουργείται και προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο και αποθηκεύεται στο πεδίο του `data` η έκφραση `"objectref.field = value;"`.

Συνάρτηση `putStaticAction`

Αυτή η συνάρτηση χειρίζεται το `bytecode` `putstatic`. Αυτό το `bytecode` αναθέτει μια τιμή σε ένα στατικό πεδίο μιας κλάσης απωθώντας από τη στοίβα τελεστών τη νέα τιμή που θα πάρει, έστω `value`. Το πεδίο προς ενημέρωση προκύπτει από την εντολή και έστω ότι η ονομασία του είναι `field`. Από την εντολή προκύπτει επίσης και το όνομα της κλάσης στην οποία ανήκει το πεδίο και έστω ότι είναι `className`. Αν για παράδειγμα έχουμε την εντολή `"putstatic #3 = Field Item.uniqueId(I)"`, τότε το πεδίο προς ενημέρωση είναι το `uniqueId` της κλάσης `Item`. Επίσης η εντολή μας πληροφορεί ότι το πεδίο είναι τύπου `int` (ο τύπος `I`, που μεταφράζεται ως `int`, περιλαμβάνεται σε ένα ζεύγος παρενθέσεων). Κατά τη διάρκεια της ανάλυσης του `bytecode` `putStatic` δημιουργείται και προστίθεται ένας νέος κόμβος στο

συντακτικό δέντρο. Ο αλγόριθμος ελέγχει αν η κλάση στην οποία ανήκει το στατικό πεδίο προς ενημέρωση, είναι ίδια με την κλάση η οποία πραγματοποιεί την αλλαγή της τιμής του πεδίου. Αν οι δυο κλάσεις είναι ίδιες τότε αποθηκεύεται στο πεδίο data του κόμβου η έκφραση “field = value;”. Σε αντίθετη περίπτωση αποθηκεύεται η έκφραση “className.field = value;”

Συνάρτηση `getStaticAction`

Αυτή η συνάρτηση χειρίζεται το bytecode `getstatic`. Αυτό το bytecode ωθεί στη στοίβα τελεστών την τιμή ενός στατικού πεδίου. Το όνομα του πεδίου και η κλάση στην οποία ανήκει προκύπτουν από την εντολή και έστω ότι είναι `field` και `className` αντίστοιχα. Αν η εντολή για παράδειγμα είναι “`getstatic #3 = Field Item.uniqueId(I)`”, τότε το πεδίο είναι το `Field` και η κλάση είναι η `Item`. Επίσης η εντολή μας πληροφορεί ότι το πεδίο είναι τύπου `int` (ο τύπος `I`, που μεταφράζεται ως `int`, περιλαμβάνεται σε ένα ζεύγος παρενθέσεων). Τέλος, ο αλγόριθμος ελέγχει αν η κλάση στην οποία ανήκει το στατικό πεδίο, είναι ίδια με την κλάση η οποία πραγματοποιεί την ώθηση της τιμής του πεδίου. Αν οι δυο κλάσεις είναι ίδιες τότε ωθείται στην στοίβα τελεστών η έκφραση “`field`”. Σε αντίθετη περίπτωση ωθείται η έκφραση “`className.field`”.

Συνάρτηση `checkcastAction`

Αυτή η συνάρτηση χειρίζεται το bytecode `checkcast`. Αυτό το bytecode ελέγχει αν ένα αντικείμενο είναι ενός δοσμένου τύπου. Η αναφορά στο αντικείμενο, έστω `ref`, απωθείται από τη στοίβα τελεστών. Ο τύπος προκύπτει από την ίδια την εντολή και έστω ότι είναι `className`. Αν για παράδειγμα έχουμε την εντολή “`checkcast java.lang.String`”, τότε ο τύπος είναι `String`. Στη συνέχεια ωθείται στη στοίβα τελεστών η έκφραση “`(className)ref`”.

Συνάρτηση `invokeAction`

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: `invokevirtual`, `invokestatic`, `invokeinterface`, `invokespecial`. Αυτά τα bytecodes καλούν μια απλή μέθοδο, μια στατική μέθοδο, μια μέθοδο μιας διεπαφής ή μια ειδική μέθοδο μιας κλάσης, όπως είναι ο `constructor`, αντίστοιχα. Στην ενέργεια `invoke`, το σύμβολο `symbol` αντιπροσωπεύει το είδος της μεθόδου που θα κληθεί. Ο τύπος των ορισμάτων της μεθόδου περιλαμβάνεται σε ένα ζεύγος παρενθέσεων στην ίδια την εντολή. Από την παράθεση των τύπων των ορισμάτων προκύπτει και το πλήθος τους. Ο τύπος επιστροφής της μεθόδου παρατίθεται επίσης στην εντολή αμέσως μετά τα ορίσματα. Για παράδειγμα, αν έχουμε την εντολή “`invokevirtual #7 = Method java.lang.Thread.start((Ljava/lang/String;I)V)`” βλέπουμε ότι το όνομα της μεθόδου είναι `start` και ότι δέχεται δυο ορίσματα, ένα `String` και έναν αχέραιο αριθμό. Ο τύπος επιστρο-

φής είναι void. Αν πάλι έχουμε για παράδειγμα την εντολή `invokespecial #4 = Method client.Client.<init>((Ljava/lang/String;I)V)` βλέπουμε ότι καλείται ένας constructor (φαίνεται από τη λέξη `<init>`) και ότι δέχεται δυο ορίσματα, ένα String και έναν ακέραιο αριθμό. Ο τύπος επιστροφής είναι void. Ο αλγόριθμος εκμαιεύει το απλό όνομα της μεθόδου από το πλήρες της όνομα, έστω `method`, και απωθεί από τη στοίβα δεδομένων τα ορίσματα, έστω $arg_1, arg_2, \dots, arg_n$. Αν η εντολή είναι η `invokevirtual`, τότε απωθείται από τη στοίβα τελεστών και η αναφορά στο αντικείμενο το οποίο καλεί τη μέθοδο, έστω `ref`. Αν όμως η εντολή είναι η `invokestatic` ή η `invokeinterface` ή η `invokespecial`, τότε εκμαιεύεται από την ίδια την εντολή το όνομα της κλάσης η οποία καλεί τη μέθοδο, έστω `className`. Κατά τη διάρκεια της ανάλυσης του bytecode δημιουργείται και προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο. Αν η μέθοδος που καλείται δεν επιστρέφει τίποτα (τύπος επιστροφής void), τότε αποθηκεύεται στο πεδίο data του νέου κόμβου στην περίπτωση της εντολής `invokevirtual` η έκφραση `ref.method(arg_1, arg_2, \dots, arg_n)`; και στην περίπτωση των εντολών `invokestatic` και `invokeinterface`, αποθηκεύεται στο πεδίο data του νέου κόμβου η έκφραση `className.method(arg_1, arg_2, \dots, arg_n)`; . Αν ο τύπος επιστροφής είναι διάφορος του void, τότε αυτό σημαίνει ότι το αποτέλεσμα που επιστρέφει η μέθοδος θα ανατεθεί σε ένα πεδίο αντικειμένου ή σε μια απλή μεταβλητή άρα οι εντολές που προαναφέρθηκαν αντί να αποθηκευτούν στο πεδίο data του νέου κόμβου, ωθούνται στη στοίβα τελεστών. Στην περίπτωση του bytecode `invokespecial`, η αναφορά στο αντικείμενο το οποίο καλεί την ειδική μέθοδο απωθείται από τη στοίβα τελεστών. Αν η αναφορά αυτή είναι η `this` τότε αποθηκεύεται στο πεδίο data του κόμβου η έκφραση `super(arg_1, arg_2, \dots, arg_n)`; διότι η ύπαρξη της αναφοράς `this` σημαίνει την κλήση ειδικής μεθόδου της υπερκλάσης. Αν η αναφορά αυτή δεν είναι η `this`, αλλά έχει τιμή έστω `ref`, τότε ωθείται στη στοίβα τελεστών η έκφραση `ref.(arg_1, arg_2, \dots, arg_n)`.

Συνάρτηση ldcAction

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: `ldc`, `ldc_w`, `ldc2_w`. Αυτά τα bytecodes ωθούν στη στοίβα τελεστών αλφαριθμητικά δεδομένα που αντλούν από το constant pool. Για παράδειγμα η εντολή `ldc #2 = "abc"` θα έχει ως αποτέλεσμα την ώθηση του String "abc" στη στοίβα τελεστών. Σε γενικές γραμμές, η σταθερά που ακολουθεί τον χαρακτήρα = στην εντολή, ωθείται στη στοίβα.

Συνάρτηση dupAction

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: `dup`, `dup2`, `dup_x1`, `dup_x2`, `dup2_x1`, `dup2_x2`. Το bytecode `dup` αντιγράφει την τιμή της κορυφής της στοίβας τελεστών και

την τοποθετεί στην κορυφή της. Το bytecode `dup2` αντιγράφει τη μία ή τις δύο τιμές που βρίσκονται πάνω πάνω στη στοίβα τελεστών και τις τοποθετεί στην κορυφή της. Το bytecode `dup_x1` αντιγράφει την τιμή της κορυφής της στοίβας τελεστών και την τοποθετεί δύο θέσεις κάτω. Το bytecode `dup_x2` αντιγράφει την τιμή της κορυφής της στοίβας τελεστών και την τοποθετεί δύο ή τρεις θέσεις κάτω. Το bytecode `dup2_x1` αντιγράφει τη μία ή τις δύο τιμές που βρίσκονται πάνω πάνω στη στοίβα τελεστών και τις τοποθετεί δύο ή τρεις θέσεις κάτω. Το bytecode `dup2_x2` αντιγράφει τη μία ή τις δύο τιμές που βρίσκονται πάνω πάνω στη στοίβα τελεστών και τις τοποθετεί δύο, τρεις ή τέσσερις θέσεις κάτω. Το σύμβολο εδώ αντιπροσωπεύει ποια από τις προαναφερθείσες διαδικασίες θα πραγματοποιηθεί.

Συνάρτηση `synchronizeAction`

Αυτή η συνάρτηση χειρίζεται το bytecode `monitorenter`. Αυτό το bytecode θέτει ένα αντικείμενο σε λειτουργία παρακολούθησης (`monitor`). Επομένως απωθείται από τη στοίβα τελεστών η αναφορά στο αντικείμενο που θα τεθεί υπό παρακολούθηση, έστω `ref` και αποθηκεύεται στο συντακτικό δέντρο η έκφραση `"synchronized(ref){"` η οποία σημαίνει την αρχή της παρακολούθησης.

Συνάρτηση `desynchronizeAction`

Αυτή η συνάρτηση χειρίζεται το bytecode `monitorexit`. Αυτό το bytecode σταματά τη διαδικασία παρακολούθησης ενός αντικειμένου (`monitor`). Η αναφορά στο αντικείμενο απωθείται από τη στοίβα τελεστών. Κατά τη διάρκεια της ανάλυσης του bytecode `monitorexit` δημιουργείται και προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο και αποθηκεύεται στο πεδίο του `data` το σύμβολο `'}'` το οποίο σημαίνει το τέλος της παρακολούθησης.

Συνάρτηση `switchAction`

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: `tableswitch`, `lookupswitch`. Τα δύο αυτά bytecodes υλοποιούν τη δομή `switch`. Η μεταβλητή η οποία θα συγκριθεί με τις περιπτώσεις απωθείται από τη στοίβα τελεστών και έστω ότι η ονομασία της είναι `var`.

Παρακάτω βλέπουμε ένα παράδειγμα δομής `switch`.

```
switch(var1){
    case 0:
        var2 = 0;
        break;
    case 1:
```

```
    var2 = 1;
    break;
case 2:
    var2 = 2;
    break;
default:
    var2 = -1;
}
return var2;
```

Παρακάτω παρατίθενται οι εντολές bytecode που αντιστοιχούν στο παραπάνω παράδειγμα.

```
2 iload_1
3 tableswitch {
    default: 43
    0: 28
    1: 33
    2: 38}
28 iconst_0
29 istore_2
30 goto 45
33 iconst_1
34 istore_2
35 goto 45
38 iconst_2
39 istore_2
40 goto 45
43 iconst_m1
44 istore_2
45 iload_2
46 ireturn
```

Το τμήμα εντολών που βρίσκεται εντός των αγκυλών αμέσως μετά τη λέξη tableswitch δίνει πληροφορίες σχετικά με τα cases. Συγκεκριμένα, η περίπτωση (case) 0 ξεκινά από την εντολή με program counter 28 και τελειώνει στην εντολή με μετρητή 30. Η περίπτωση 1 ξεκινά από την εντολή με program counter 33 και τελειώνει στην εντολή με μετρητή 35. Παρομοίως

η περίπτωση 2 ξεκινά από την εντολή με program counter 38 και τελειώνει στην εντολή με μετρητή 40. Τέλος, η default περίπτωση ξεκινά από την εντολή με program counter 43 και τελειώνει στην εντολή με μετρητή 44. Όπως παρατηρούμε, η τελευταία εντολή κάθε περίπτωσης είναι η goto 45. Η εντολή αυτή υλοποιεί το break που σημαίνει έξοδο από τη δομή switch, διότι μεταφέρει την εκτέλεση του προγράμματος στην εντολή που έπεται της δομής switch που στη συγκεκριμένη περίπτωση είναι η εντολή με μετρητή 45 η οποία φορτώνει στη στοίβα τελεστών τον ακέραιο αριθμό 2. Συνεπώς ο αλγόριθμος δημιουργεί έναν νέο κόμβο στο συντακτικό δέντρο και αποθηκεύει στο πεδίο του data την έκφραση “switch(var){”, και για κάθε περίπτωση διάφορη της default που έστω ότι έχει την ονομασία newCase, δημιουργεί επίσης έναν νέο κόμβο και αποθηκεύει στο πεδίο του data την έκφραση “case newCase:” ακολουθούμενη από τον αντίστοιχο κώδικα της. Επιπλέον, κάθε φορά που συναντάται η εντολή goto μέσα σε κάποια περίπτωση δημιουργείται ένας νέος κόμβος και αποθηκεύεται στο πεδίο του data η εντολή “break;” Αν η περίπτωση είναι η default, τότε ο αλγόριθμος δημιουργεί ξανά έναν νέο κόμβο και αποθηκεύει σε αυτόν την έκφραση “default:” ακολουθούμενη από τον αντίστοιχο κώδικα της. Όταν αποθηκεύσει όλες τις περιπτώσεις, ο αλγόριθμος κλείνει τη δομή δημιουργώντας έναν νέο κόμβο και αποθηκεύοντας σε αυτόν το σύμβολο ‘}’ το οποίο δηλώνει και το τέλος της δομής.

Συνάρτηση branchAction

Αυτή η συνάρτηση χειρίζεται τα εξής bytecodes: if_icmpeq, if_icmpne, if_icmplt, if_icmpgt, if_icmple, if_icmpge, ifeq, ifne, iflt, ifgt, ifge, ifle, ifnonnull, ifnull, if_acmpeq, if_acmpne, fcmpl, fcmpg, dcmpl, dcmpg, lcmp, goto, goto_w. Οι εντολές αυτές (με εξαίρεση τα bytecodes goto και goto_w τα οποία μεταφέρουν τη ροή του προγράμματος σε άλλη εντολή) είναι υπεύθυνες για τη δημιουργία των υποθετικών και επαναληπτικών δομών. Ένα παράδειγμα μιας τέτοιας εντολής είναι η εντολή ifnull 10 η οποία μας λέει ότι αν η αναφορά που βρίσκεται στην κορυφή της στοίβας τελεστών είναι null τότε η εκτέλεση θα συνεχίσει από την εντολή με program counter ίσο με 10. Σε μια τέτοια εντολή πάντα υπάρχει ο program counter της εντολής από την οποία θα συνεχίσει η εκτέλεση του προγράμματος αν ισχύει η υπόθεση. Επειδή σε ένα πρόγραμμα μπορεί να υπάρχουν πολλές δομές είτε υποθετικές είτε επαναληπτικές, για να γίνεται κατανοητό που αρχίζει η μια και που τελειώνει η άλλη, υπάρχει μια στοίβα που αποθηκεύει τους κόμβους του συντακτικού δέντρου που περιέχουν εντολές σύγκρισης ή επανάληψης. Η στοίβα αυτή ονομάζεται (branchNodeStack). Μέσω της στοίβας αυτής, όταν προστίθεται στο συντακτικό δέντρο ένας νέος κόμβος μέσω της συνάρτησης addNode της κλάσης AbstractSyntaxTree, γίνονται και όλες οι απαραίτητες συγκρίσεις στις τιμές των

program counters (πεδίο pcValue του κόμβου) ώστε να γίνει κατανοητό αν υπάρχει εμφώλευση και τι είδους. Στην περίπτωση του branchAction το σύμβολο symbol αντιπροσωπεύει το είδος της σύγκρισης που θα λάβει χώρα (για παράδειγμα ==, !=, <, >, <=, >=, ==0, !=0, <0, >0, <=0, >=0, !=null, ==null)

Παρακάτω βλέπουμε πως μια δομή if μεταφράζεται σε bytecodes.

```
if(var2!=null){
    var1 = 1;
}
return 1;
```

Συγκεκριμένα βλέπουμε πως αν η μεταβλητή var2 είναι null τότε δεν εκτελούνται οι εντολές με program counter 8 και 9 αλλά η εκτέλεση συνεχίζεται κατευθείαν από την εντολή με μετρητή 10. Αν όμως η μεταβλητή είναι ίση με null τότε θα εκτελεστούν όλες οι εντολές συμπεριλαμβανομένων και αυτών με μετρητή 8 και 9.

```
5 ifnull 10
8 iconst_1
9 istore_1
10 iconst_1
11 ireturn
```

Παρακάτω βλέπουμε πως μια δομή if - else if - else μεταφράζεται σε bytecodes.

```
if(var1 == 0 ){
    var2 = 1;
}
else if(9 == var1){
    var2 = 0;
}
else{
    var2 = 4;
}
return 1;
```

Συγκεκριμένα βλέπουμε ότι αν η μεταβλητή var1 δεν είναι ίση με το 0, τότε η εκτέλεση συνεχίζεται από την εντολή με μετρητή 13, η οποία ελέγχει αν η var1 είναι ίση με 9 και αν δεν είναι, η εκτέλεση προχωρά από την εντολή με μετρητή 24 η οποία ουσιαστικά υλοποιεί το

else αφού είναι οι εντολές που εκτελούνται μόνο αν δεν ισχύουν ούτε το if ούτε το else if. Αν ισχύει όμως το if, τότε εκτελούνται οι εντολές του και η εκτέλεση μεταφέρεται με μια εντολή goto στις εντολές που βρίσκονται αμέσως μετά τη δομή if - else if - else. Το ίδιο γίνεται και σε περίπτωση που ισχύει το else if. Συμπερασματικά αν στα bytecodes έχουμε συνεχόμενες εντολές σύγκρισης που αν δεν ισχύει η μια η εκτέλεση του προγράμματος οδηγείται στην επόμενη εντολή σύγκρισης, και αν ενδιάμεσα από κάθε εντολή σύγκρισης υπάρχει εντολή goto που οδηγεί πάντα στην ίδια τιμή program counter που βρίσκεται εκτός της υποθετικής δομής, όπως εδώ το goto 26, τότε σίγουρα υπάρχει δομή if - else if - else.

```
4 iload_1
5 ifne 13
8 iconst_1
9 istore_2
10 goto 26
13 iload_1
14 bipush 9
16 if_icmpne 24
19 iconst_0
20 istore_2
21 goto 26
24 iconst_4
25 istore_2
26 iconst_1
27 ireturn
```

Παρακάτω βλέπουμε πως μια δομή if - else μεταφράζεται σε bytecodes.

```
if(var2!=null){
    var1 = 1;
}
else{
    var1 = 0;
}
return 1;
```

Συγκεκριμένα βλέπουμε ότι υπάρχει μόνο μια εντολή σύγκρισης που οδηγεί σε εντολή με

μετρητή μεγαλύτερο από αυτόν της τρέχουσας. Ακριβώς πριν από την εντολή που οδηγεί η εντολή σύγκρισης `ifnull`, υπάρχει εντολή `goto` που οδηγεί σε εντολή με ακόμα μεγαλύτερο μετρητή (`goto 18`). Δηλαδή, αν η `var2` είναι `null`, τότε η εκτέλεση συνεχίζεται από την εντολή με μετρητή 16 όπου είναι ουσιαστικά εντολή που περιλαμβάνεται στο `else`. Αν όμως η μεταβλητή είναι `null`, τότε θα εκτελεστούν οι εντολές του `if` και η εντολή `goto 18` θα οδηγήσει έξω από τη δομή `if - else`. Παρόμοιες ακολουθίες εντολών `bytecodes` σηματοδοτούν την ύπαρξη δομής `if - else`.

```
8 ifnull 16
11 iconst_1
12 istore_1
13 goto 18
16 iconst_0
17 istore_1
18 iconst_1
19 ireturn
```

Όταν λοιπόν ο αλγόριθμος αναγνωρίσει μια από τις παραπάνω δομές, δημιουργεί έναν νέο κόμβο του συναντικού δέντρου καλώντας την συνάρτηση `setCurrentIfNode` η οποία αναλαμβάνει να συνδέσει τον κόμβο στο συντακτικό δέντρο. Στη συνέχεια καλείται η συνάρτηση `setIfNodeData` η οποία αναλαμβάνει να αποθηκεύσει τον πηγαίο κώδικα που αντιστοιχεί στην τρέχουσα εντολή `bytecode` στο πεδίο `data` του κόμβου.

Παρακάτω βλέπουμε πως μια δομή `while` μεταφράζεται σε `bytecodes`.

```
while(100 > var1){
    var2 = 1;
    var1 += 1;
}
var2 = 3;
```

Συγκεκριμένα βλέπουμε ότι υπάρχει μια σύγκριση της μεταβλητής `var1` με τον αριθμό 100. Όπως είναι ήδη γνωστό, ο έλεγχος για την παραμονή σε ένα βρόγχο γίνεται με σύγκριση. Έτσι γίνεται λοιπόν και εδώ. Όταν η μεταβλητή `var1` γίνει μεγαλύτερη ή ίση του 100 τότε θα γίνει έξοδος από το βρόγχο και η εκτέλεση θα συνεχιστεί από την εντολή με μετρητή 18, η οποία είναι η ώθηση του αριθμού 3 στη στοίβα τελεστών. Κάθε φορά που εκτελείται το περιεχόμενο του βρόγχου ο έλεγχος του προγράμματος πρέπει να μεταφερθεί πάλι στην

ίδια σύγκριση για να διαπιστωθεί αν θα εκτελεστεί πάλι το περιεχόμενο του ή όχι. Αυτό επιτυγχάνεται με την εντολή `goto 4` που υπάρχει στο τέλος του βρόγχου. Η εντολή αυτή οδηγεί στη σύγκριση της μεταβλητής `var1` με τον αριθμό 100. Ουσιαστικά οδηγεί σε εντολή με `program counter` μικρότερο από αυτόν της τρέχουσας και αυτό είναι ένα στοιχείο που δείχνει με απόλυτη σιγουριά ότι υπάρχει δομή επανάληψης και όχι απλή υποθετική δομή. Επιπρόσθετα, τονίζεται ότι όταν υπάρχει εντολή `goto` η οποία οδηγεί σε εντολή με μικρότερο `program counter`, τότε σίγουρα υπάρχει δομή `while`.

```
4 iload_1
5 bipush 100
7 if_icmpge 18
10 iconst_1
11 istore_2
12 iinc 1
15 goto 4
18 iconst_3
19 istore_2
```

Όταν λοιπόν ο αλγόριθμος αναγνωρίσει μια δομή `while`, δημιουργεί έναν νέο κόμβο, τον συνδέει στο συντακτικό δέντρο και αποθηκεύει τον πηγαίο κώδικα που αντιστοιχεί στην τρέχουσα εντολή `bytecode` στο πεδίο του `data`.

Παρακάτω βλέπουμε πως μια δομή `do - while` μεταφράζεται σε `bytecodes`.

```
do{
    int var2 = 1;
    var1+= 1;
}while(100 >var1)
```

Συγκεκριμένα βλέπουμε ότι η εντολή `bytecode` η οποία εκτελεί τη σύγκριση για την παραμονή στο βρόγχο (`if_icmplt 2`) βρίσκεται στο τέλος και όχι στην αρχή όπως στη δομή `while` και εκτός αυτού οδηγεί σε εντολή με `program counter` μικρότερο από αυτόν της τρέχουσας εντολής αφού ουσιαστικά οδηγεί ξανά στο περιεχόμενο του βρόγχου αν έχει αποφασιστεί η παραμονή σε αυτόν, αν δηλαδή η μεταβλητή `var1` συνεχίσει να είναι μικρότερη του 100. Από αυτό φαίνεται και το γεγονός ότι οι εντολές που βρίσκονται εντός ενός βρόγχου `do - while` εκτελούνται τουλάχιστον μια φορά αφού η σύγκριση βρίσκεται στο τέλος. Όταν λοιπόν έχουμε εντολή σύγκρισης που οδηγεί σε εντολή με μικρότερο `program counter` τότε είναι

σίγουρο ότι υπάρχει δομή do - while.

```
2 iconst_1
3 istore_2
4 iinc 1
7 iload_1
8 bipush 100
10 if_icmplt 2
```

Όταν λοιπόν ο αλγόριθμος αναγνωρίσει μια δομή do - while, δημιουργεί έναν νέο κόμβο του συναντικού δέντρου καλώντας την συνάρτηση `createLoopNode` η οποία αναλαμβάνει να συνδέσει τον κόμβο στο συντακτικό δέντρο. Στη συνέχεια καλείται η συνάρτηση `setDoWhileNodeData` η οποία αναλαμβάνει να αποθηκεύσει τον πηγαίο κώδικα που αντιστοιχεί στην τρέχουσα εντολή bytecode στο πεδίο `data` του κόμβου.

Σημειώνεται ότι οι τιμές που απαιτούνται για να γίνει η σύγκριση απωθούνται από τη στοίβα τελεστών.

Περνώντας τώρα στα bytecodes `goto` και `goto_w` σημειώνεται ότι η δράση τους είναι να μεταφέρεται η εκτέλεση του προγράμματος στην εντολή με program counter ίσο με τον αριθμό που παρατίθεται ακριβώς δίπλα τους. Αν για παράδειγμα, η εντολή είναι η `goto 50`, τότε η εκτέλεση μεταφέρεται στην εντολή με μετρητή 50. Η εντολή αυτή μόνη της δεν έχει νόημα. Συμπεριλαμβάνεται μόνο σε υποθετικές ή επαναληπτικές δομές (`if`, `while`, `do-while`, `switch`). Ο τρόπος χειρισμού της εντολής αυτής από τον αλγόριθμο έχει περιγραφεί στην παραπάνω ανάλυση των εξής bytecodes: `if_icmpeq`, `if_icmpne`, `if_icmplt`, `if_icmpgt`, `if_icmple`, `if_icmpge`, `ifeq`, `ifne`, `iflt`, `ifgt`, `ifge`, `ifle`, `ifnonnull`, `ifnull`, `if_acmpeq`, `if_acmpne`, `fcmpl`, `fcmpg`, `dcmpl`, `dcmpg`, `lcmp`, `tableswitch`, `lookupswitch`. [3][12]. Αξίζει να σημειωθεί ο τρόπος λειτουργίας όμως του αλγορίθμου σε περίπτωση που η τιμή του program counter στον οποίο οδηγεί η εντολή `goto` είναι μεγαλύτερη της τιμής του τρέχοντος program counter. Ο αλγόριθμος στην περίπτωση αυτή πραγματοποιεί ελέγχους στην λίστα `caseNodeList`, η οποία έχει αποθηκευμένες τις περιπτώσεις των δομών `switch` που έχουν εντοπιστεί, προκειμένου να εξακριβωθεί αν η εντολή `goto` ανήκει σε μια τέτοια δομή. Εάν η εντολή `goto` αποτελεί λοιπόν μέρος μιας τέτοιας δομής, τότε ελέγχεται αν πρόκειται για την περίπτωση `default`. Αν διαπιστωθεί ότι πρόκειται για την περίπτωση `default`, τότε προστίθενται τρεις νέοι κόμβοι στο συντακτικό δέντρο: στον πρώτο αποθηκεύεται ως δεδομένο η έκφραση `"default:{"`, στον δεύτερο ο πηγαίος κώδικας που αντιστοιχεί στην περίπτωση `default`, ο οποίος είναι ήδη αποθηκευμένος

στη λίστα `caseNodeList`, και στον τρίτο το σύμβολο `'}` το οποίο σηματοδοτεί το τέλος της περίπτωσης `default`. Σε περίπτωση που η εντολή `goto` ανήκει σε δομή `switch` αλλά δεν πρόκειται για την περίπτωση `default`, τότε αντιστοιχεί στην εντολή `break`. Συνεπώς προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο και αποθηκεύεται σε αυτόν η έκφραση `"break;"`.

Στην περίπτωση που ο αλγόριθμος αποκλείσει το γεγονός η εντολή `goto` να είναι μέρος μιας δομής `switch`, τότε η εντολή αντιστοιχεί είτε σε μια έκφραση `else if` είτε σε μια έκφραση `else`. Ο αλγόριθμος εκμαιεύει σε ποιά από τις δύο εκφράσεις αντιστοιχεί, πραγματοποιώντας ελέγχους στην στοίβα `gotoNodeStack`, η οποία έχει αποθηκευμένες τις εντολές `goto`. Στην περίπτωση `else if`, ο κόμβος με τα δεδομένα υπάρχει ήδη στο δέντρο αλλά έχει αποθηκευτεί σαν μια εντολή `if` και αυτό διότι η εντολή `goto` βρίσκεται παρακάτω από τις εντολές που αντιστοιχούν στην σύγκριση και έτσι ο αλγόριθμος δεν μπορεί να γνωρίζει αν πρόκειται για εντολή `if` ή `else if`. Ο αλγόριθμος λοιπόν, αναζητά και βρίσκει στο συντακτικό δέντρο τον τελευταίο κόμβο `if` και αντικαθιστά στα αποθηκευμένα δεδομένα του την έκφραση `if` με την έκφραση `else if`. Επιπλέον προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο και αποθηκεύεται σε αυτόν το σύμβολο `'}` το οποίο σηματοδοτεί το τέλος της περίπτωσης `else if`. Τέλος, στην περίπτωση που η εντολή `goto` αντιστοιχεί στην έκφραση `else`, τότε προστίθεται ένας νέος κόμβος στο συντακτικό δέντρο και αποθηκεύεται σε αυτόν η έκφραση `"else{"`. Ο πηγαίος κώδικας που αντιστοιχεί στην περίπτωση `else` βρίσκεται στις επόμενες εντολές `bytecode`.

Συνάρτηση `setCurrentDoWhileNode`

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `branchAction` όταν αναγνωριστεί από τον αλγόριθμο ότι υπάρχει μια δομή `do-while`. Συγκεκριμένα, η συνάρτηση αναλαμβάνει να εντοπίσει την θέση στην οποία θα τοποθετηθεί ο κόμβος του συντακτικού δέντρου ο οποίος θα περιέχει την έκφραση `"do{"` η οποία σηματοδοτεί την αρχή της δομής `do-while`.

Συνάρτηση `createLoopNode`

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `branchAction` όταν αναγνωριστεί από τον αλγόριθμο ότι υπάρχει μια δομή `do-while`. Συγκεκριμένα, η συνάρτηση αναλαμβάνει να δημιουργήσει και να τοποθετήσει στο συντακτικό δέντρο τον νέο κόμβο που αντιστοιχεί στην έναρξη της δομής `do-while`. Ο νέος αυτός κόμβος περιέχει ως δεδομένο την έκφραση `"do{"` η οποία σηματοδοτεί την αρχή μιας δομής `do-while`.

Συνάρτηση `setDoWhileNodeData`

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `branchAction`

όταν αναγνωριστεί από τον αλγόριθμο ότι υπάρχει μια δομή do-while. Συγκεκριμένα, η συνάρτηση αποθηκεύει στον κόμβο του συντακτικού δέντρου που ανήκει στην συνθήκη τερματισμού της δομής do-while, τον αντίστοιχο πηγαίο κώδικα.

Συνάρτηση findDoWhileNode

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση createLoopNode όταν χρειαστεί να εντοπιστεί στο συντακτικό δέντρο ένας ήδη υπάρχων κόμβος ο οποίος ύστερα από ελέγχους προκύπτει ότι αυτός δεν ανήκει τελικά σε δομή do-while και θα πρέπει συνεπώς να αλλαχθούν τα αποθηκευμένα δεδομένα του.

Συνάρτηση setCurrentIfNode

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση branchAction όταν αναγνωριστεί από τον αλγόριθμο ότι υπάρχει μια δομή υπόθεσης. Η συνάρτηση αναλαμβάνει να συνδέσει τον νέο κόμβο στο συντακτικό δέντρο. Η συνάρτηση καλείται επίσης και όταν ο αλγόριθμος εντοπίζει ότι υπάρχει μια συνθήκη η οποία πιθανόν να ανήκει σε μια ήδη υπάρχουσα δομή if. Συγκεκριμένα, η συνάρτηση αναλαμβάνει να εντοπίσει τον κόμβο στον οποίο θα πρέπει να αποθηκευτεί η νέα συνθήκη. Σημειώνεται ότι ο κόμβος αυτός περιέχει ήδη μια συνθήκη if επομένως η νέα συνθήκη θα προστεθεί με την χρήση ενός λογικού τελεστή.

Συνάρτηση branchNodeConcatenation

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από τις συναρτήσεις setCurrentIfNode και setCurrentDoWhileNode και ο ρόλος της είναι να συνενώσει δύο κόμβους του συντακτικού δέντρου που περιέχουν αποθηκευμένες συνθήκες if ή do-while αντίστοιχα. Ο αλγόριθμος έχει εντοπίσει ότι αυτές οι συνθήκες ανήκουν στην ίδια εντολή υπόθεσης ή επανάληψης οπότε η συνάρτηση αναλαμβάνει τη συνένωση τους.

Συνάρτηση containsGotoStatement

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση setCurrentIfNode. Ο ρόλος αυτής της συνάρτησης είναι να εντοπίσει αν στις εντολές bytecode υπάρχει η εντολή goto προκειμένου να εντοπιστούν πιθανές δομές υπόθεσης ή περιπτώσεις στις οποίες θα πρέπει να γίνει συνένωση κόμβων που περιέχουν αποθηκευμένες συνθήκες if.

Συνάρτηση inverseRelationalOperator

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από τις συναρτήσεις setIfNodeData και branchAction και ο ρόλος της είναι να αντιστρέψει τους σχεσιακούς τελεστές. Για παράδειγμα ο τελεστής '==' μετετρέπεται σε '!=' ενώ ο τελεστής '!=' σε '=='. Αυτή η συνάρτηση

χρειάζεται γιατί οι εντολές bytecode που αντιστοιχούν σε δομές υπόθεσης έχουν ως τελεστή τον αντίστροφο από αυτόν που έχουν οι εντολές if που τους αντιστοιχούν. Για παράδειγμα στο παρακάτω απόσπασμα κώδικα, η εντολή bytecode “ifnull 10”, μας λέει ότι αν η τιμή της μεταβλητής που θα αποθηκευθεί από την στοίβα τελεστών είναι null τότε οι εντολές με μετρητή 8 και 9 θα παραλειφθούν και η εκτέλεση θα συνεχιστεί από την εντολή με μετρητή 10. Η αντίστοιχη εντολή if είναι “if(var2!=null){”. Δηλαδή αν η τιμή της μεταβλητής var2 είναι ίση με null τότε θα εκτελεστούν οι εντολές με μετρητή 8 και 9, σε διαφορετική περίπτωση η εκτέλεση θα συνεχιστεί από την εντολή με μετρητή 10. Η συνθήκη δηλαδή σε μια εντολή υπόθεσης bytecode, δείχνει πότε δεν θα εκτελεστούν οι εντολές που βρίσκονται εντός του block υπόθεσης. Αντιθέτως, μια συνθήκη υπόθεσης στον πηγαίο κώδικα, δείχνει πότε θα εκτελεστούν οι εντολές που βρίσκονται μέσα στο block της υπόθεσης. Για αυτόν τον λόγο χρειάζεται και η αντιστροφή των σχεσιακών τελεστών.

Παράδειγμα εντολών bytecode που αντιστοιχούν σε μια δομή υπόθεσης:

```
5 ifnull 10
8 iconst_1
9 istore_1
10 iconst_1
11 ireturn
```

Ο πηγαίος κώδικας που αντιστοιχεί στις παραπάνω εντολές bytecode:

```
if(var2!=null){
    var1 = 1;
}
return 1;
```

Συνάρτηση setIfNodeData

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από τις συναρτήσεις branchNodeConcatenation και branchAction. Ο ρόλος αυτής της συνάρτησης είναι να αποθηκεύσει τα δεδομένα στον κόμβο του συντακτικού δέντρου που αντιπροσωπεύει τη συνθήκη της δομής υπόθεσης. Συγκεκριμένα, η συνάρτηση είτε αποθηκεύει στο πεδίο data του κόμβου την έκφραση ‘if(συνθήκη){’ αν έχει μόλις εντοπιστεί μια νέα δομή υπόθεσης, είτε συμπληρώνει μια ήδη υπάρχουσα συνθήκη με επιπλέον κριτήρια ελέγχου. Στην δεύτερη περίπτωση ο αλγόριθμος έχει εντοπίσει ότι στην συνθήκη ελέγχου της δομής αντιστοιχούν περισσότερα από ένα κριτήρια τα οποία συνενώνονται με έναν σχεσιακό τελεστή όπως για παράδειγμα στην

έκφραση “if(var2!=null && var3!=null){”.

Συνάρτηση calculatePC

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από τη συνάρτηση printMethod. Αυτή η συνάρτηση επιστρέφει για κάθε εντολή bytecode μια ακέραια τιμή η οποία αντιστοιχεί στον αποθηκευτικό χώρο που καταλαμβάνει (με μονάδα μέτρησης το byte). Με αυτόν τον τρόπο, η μέθοδος printMethod είναι σε θέση να υπολογίσει τον program counter που αντιστοιχεί σε κάθε εντολή bytecode ώστε να τον εμφανίσει αριστερά της. Για παράδειγμα η εντολή bytecode “anewarray” καταλαμβάνει χώρο που αντιστοιχεί σε 4 bytes ενώ η εντολή “invokevirtual” καταλαμβάνει χώρο που αντιστοιχεί σε 3 bytes.

Συνάρτηση printMethod

Αυτή η συνάρτηση είναι υπεύθυνη για την εμφάνιση του πηγαίου κώδικα μιας συνάρτησης της επιλεγμένης από τον χρήστη κλάσης στο γραφικό περιβάλλον της εφαρμογής. Η printMethod ξεκινάει εμφανίζοντας τον τύπο επιστροφής, το όνομα, τα ορίσματα και τυχόν εξαιρέσεις της συνάρτησης. Η συνάρτηση printMethod έχει πρόσβαση στις εξαιρέσεις της συνάρτησης κάνοντας χρήση της μεθόδου getExceptionsAttribute() της κλάσης MethodInfo. Ο τύπος επιστροφής καθώς και τα ορίσματα λαμβάνονται ύστερα από επεξεργασία μέσω της μεθόδου getDescriptor() της κλάσης MethodInfo. Επιπλέον, ο τύπος πρόσβασης της συνάρτησης λαμβάνεται μέσω της μεθόδου getAccessFlags() της κλάσης MethodInfo. Στη συνέχεια η printMethod λαμβάνει τις εντολές bytecode της συνάρτησης, καλώντας τη μέθοδο getCodeAttribute() της κλάσης MethodInfo. Για κάθε εντολή bytecode της συνάρτησης, η printMethod καλεί την συνάρτηση disassembler η οποία ξεκινά τη διαδικασία δημιουργίας του συντακτικού δέντρου, του οποίου τα δεδομένα εμφανίζονται στο γραφικό περιβάλλον της εφαρμογής, μέσω κλήσης της μεθόδου printTree της κλάσης AbstractSyntaxTree.

Συνάρτηση escapeCharacters

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από τη συνάρτηση ldcAction. Όπως προαναφέρθηκε στην ανάλυση της συνάρτησης ldcAction, όταν αυτή καλείται, ωθείται μια αλφαριθμητική τιμή στην στοίβα τελεστέων. Αυτή η αλφαριθμητική τιμή μπορεί να περιέχει χαρακτήρες διαφυγής (escape characters) και την περίπτωση αυτή έρχεται να καλύψει η συνάρτηση escapeCharacters καθώς οι χαρακτήρες διαφυγής απαιτούν έναν συγκεκριμένο χειρισμό. Οι χαρακτήρες διαφυγής τους οποίους χειρίζεται η συνάρτηση escapeCharacters είναι οι εξής: \, \n, \t, \r, \b, \f, \', \”. Αυτοί οι χαρακτήρες καθώς αποτελούν μέρος ενός String, θα πρέπει να αποθηκευτούν ακριβώς όπως είναι (και να μην αντικατασταθούν από τον

χαρακτήρα στον οποίο αντιστοιχούν) στην στοίβα τελεστών και επίσης να εμφανιστούν με τον ακριβώς ίδιο τρόπο στον πηγαίο κώδικα που θα εμφανίσει ο decompiler. Για να επιτευχθεί αυτό, η συνάρτηση `escapeCharacters`, όταν λάβει ως όρισμα έναν τέτοιο χαρακτήρα, του προσθέτει για κάθε backslash που περιέχει ένα επιπλέον backslash, έτσι ώστε ο αλγόριθμος να μην τον αντικαταστήσει με τον χαρακτήρα στον οποίον αντιστοιχεί αλλά να τον διατηρήσει ως έχει.

Συνάρτηση `genericsHandling`

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `invokeAction`. Αυτή η συνάρτηση χειρίζεται τους παραμετρικούς τύπους δεδομένων (generics). Οι παραμετρικοί τύποι δεδομένων ξεχωρίζουν από την τοποθέτηση τους ανάμεσα στα σύμβολα '`<`' και '`>`'. Ο αλγόριθμος αντιλαμβάνεται την ύπαρξη ενός παραμετρικού τύπου δεδομένων όταν σε μια εντολή bytecode `invokevirtual` γίνεται κλήση μιας μεθόδου μιας δομής δεδομένων η οποία ανήκει σε μια βιβλιοθήκη της Java (όπως η `util`) και υπάρχει ως όρισμα τουλάχιστον ένα αντικείμενο τύπου `Object`. Όταν λοιπόν ο αλγόριθμος εντοπίσει έναν παραμετρικό τύπο δεδομένων, καλεί την συνάρτηση `genericsHandling`. Η συνάρτηση `genericsHandling` στη συνέχεια αναλαμβάνει ύστερα από επεξεργασία των ορισμάτων της μεθόδου που καλεί η εντολή bytecode `invokevirtual`, να αποθηκεύσει στο συντακτικό δέντρο τον πηγαίο κώδικα που αντιστοιχεί στην κλήση της μεθόδου, κάνοντας χρήση των παραμετρικών τύπων δεδομένων.

Ε'.5 Το συντακτικό δέντρο

Το συντακτικό δέντρο υλοποιείται με την βοήθεια δύο κλάσεων. Η πρώτη κλάση είναι η `AbstractSyntaxTreeNode` η οποία αντιπροσωπεύει τον κόμβο του δέντρου και αποτελείται από τα εξής πεδία: ένα πεδίο `data` στο οποίο αποθηκεύεται ο πηγαίος κώδικας της εντολής που αντιστοιχεί στον κόμβο, ένα πεδίο `pcValue` το οποίο έχει αποθηκευμένο τον program counter της εντολής, και δύο δείκτες που συνδέουν τον κόμβο με τον αριστερό και δεξί του κόμβο (`left` και `right` αντίστοιχα). Η κλάση περιέχει επίσης και τους αντίστοιχους constructors καθώς και τις αντίστοιχες μεθόδους `set` και `get`. Η δεύτερη κλάση είναι η `AbstractSyntaxTree` η οποία αντιπροσωπεύει το συντακτικό δέντρο και αποτελείται από ένα πεδίο το οποίο είναι ένας δείκτης στην ρίζα του δέντρου.

Παρακάτω αναλύονται οι συναρτήσεις που φέρνουν εις πέρας τις ενέργειες που αντιστοιχούν σε κάθε bytecode:

Συνάρτηση addNode

Αυτή η συνάρτηση προσθέτει έναν νέο κόμβο στο συντακτικό δέντρο. Η τοποθέτηση του νέου κόμβου στο δέντρο γίνεται κυρίως με βάση την τιμή που έχει ο pc counter. Η διασύνδεση του κόμβου με τον δεξί και αριστερό του κόμβο γίνεται με βάση το είδος της εντολής, αν δηλαδή έχουμε δομή επανάληψης, αν έχουμε δομή επιλογής ή δομή ακολουθίας. Για παράδειγμα σε μια δομή do-while, ο δεξιός δείκτης του κόμβου με την εντολή που περιέχει την συνθήκη ελέγχου, δείχνει πίσω και συγκεκριμένα στο κόμβο που σηματοδοτεί την αρχή της δομής do-while. Σε μια συνθήκη while ο αριστερός δείκτης του κόμβου που περιέχει την δομή ελέγχου, δείχνει στον κόμβο που περιέχει την εντολή που έπεται της δομής while, ενώ ο δεξιός δείκτης δείχνει στον κόμβο που περιέχει την πρώτη εντολή της δομής while.

Συνάρτηση findNode

Αυτή η συνάρτηση αναζητά στο συντακτικό δέντρο έναν κόμβο με μια συγκεκριμένη τιμή pc counter. Η αναζήτηση πραγματοποιείται σαρώνοντας το δέντρο ξεκινώντας από τη ρίζα, μέχρι να βρεθεί κόμβος με τιμή pc counter ίση με την τιμή pcValue που έχει δοθεί σαν όρισμα στην συνάρτηση.

Συνάρτηση printTree

Αυτή η συνάρτηση τυπώνει τα περιεχόμενα των κόμβων του συντακτικού δέντρου ξεκινώντας από την ρίζα. Ο πηγαίος κώδικας ο οποίος είναι αποθηκευμένος στους κόμβους του δέντρου, εμφανίζεται με τις απαιτούμενες εσοχές. Επιπλέον, οι δεσμευμένες λέξεις εμφανίζονται με μπλε χρώμα και οι συμβολοσειρές με πορτοκαλί. Για να επιτευχθεί η σωστή σάρωση του δέντρου και συνεπώς η σωστή εμφάνιση του πηγαίου κώδικα γίνεται χρήση μιας στοίβας (nodeStack). Η στοίβα περιέχει τους κόμβους του δέντρου των οποίων το περιεχόμενο έχει ήδη εμφανιστεί αλλά έχουν δεξί ή/και αριστερό παιδί οπότε θα πρέπει αφού ακολουθήσουμε το μονοπάτι αυτών των κόμβων (παιδιών), να επιστρέψουμε σε αυτούς ώστε να συνεχιστεί από το σωστό σημείο η σάρωση του δέντρου.

Συνάρτηση popNode

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση printTree και απωθεί από τη στοίβα nodeStack που περιέχει αποθηκευμένους ορισμένους κόμβους του δέντρου, τον κόμβο που βρίσκεται στην κορυφή της.

Συνάρτηση correctBranchNodeData

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση printTree όταν

η εντολή προς εμφάνιση περιέχει μια συνθήκη υπόθεσης (π.χ. `if`, `else if`) ή μια συνθήκη επανάληψης (π.χ. `while`, `do-while`) και έχει ως στόχο να προσθέσει τυχόν παρενθέσεις που λείπουν. Ο αλγόριθμος αναγνωρίζει αν λείπει κάποια παρένθεση μετρώντας το πλήθος των αριστερών και των δεξιών παρενθέσεων και υπολογίζοντας τη διαφορά τους. Αν διαπιστωθεί ότι η διαφορά του πλήθους των αριστερών και των δεξιών παρενθέσεων δεν είναι μηδενική, τότε γίνεται διόρθωση των δεδομένων του κόμβου και προστίθενται οι παρενθέσεις που λείπουν.

Συνάρτηση `optimizeBranchNodeData`

Πρόκειται για μια βοηθητική συνάρτηση η οποία καλείται από την συνάρτηση `printTree` όταν η εντολή προς εμφάνιση περιέχει μια συνθήκη υπόθεσης (π.χ. `if`, `else if`) ή μια συνθήκη επανάληψης (π.χ. `while`, `do-while`) και έχει ως στόχο να διαγράψει περιττές παρενθέσεις ώστε να είναι πιο εμφανίσιμος ο πηγαίος κώδικας. Η διαγραφή είναι κυρίως απαραίτητη όταν έχουμε πολλαπλές συνθήκες ελέγχου σε μια εντολή υπόθεσης ή σε μια εντολή επανάληψης. Η διαγραφή των παρενθέσεων υλοποιείται με την βοήθεια μιας στοίβας `bracketStack` στην οποία τοποθετούνται οι θέσεις των αριστερών παρενθέσεων '(' . Στη συνέχεια όταν συναντάται δεξιά παρένθεση ')' με διάφορους ελέγχους διαπιστώνεται αν έχουμε πολλαπλές συνθήκες ελέγχου και αν είναι αποδεκτή η διαγραφή των παρενθέσεων. Αν η διαγραφή είναι αποδεκτή τότε σε μια δομή `HashSet` με την ονομασία `bracketsToDelete` τοποθετούνται οι θέσεις του ζεύγους των παρενθέσεων (η θέση της τρέχουσας δεξιάς παρένθεσης και η θέση της αριστερής παρένθεσης που της αντιστοιχεί η οποία είναι αποθηκευμένη στην κορυφή της στοίβας `bracketStack`). Η δομή `bracketsToDelete` στο τέλος θα περιέχει τις θέσεις των παρενθέσεων προς διαγραφή και η συνάρτηση με βάση αυτήν θα υλοποιήσει τις διαγραφές.

Κεφάλαιο Γ΄

ΠΑΡΑΔΕΙΓΜΑΤΑ

Σε όλα τα παραδείγματα δίνεται ως είσοδος στην εφαρμογή το .class αρχείο της κλάσης που αναλύεται ώστε να φαίνονται τα αποτελέσματα της λειτουργίας και του bytecode viewer αλλά και του decompiler. Επίσης σε κάθε παράδειγμα θα δίνεται και ο αυθεντικός πηγαίος κώδικας ώστε να μπορεί να γίνει σύγκριση με το τον πηγαίο κώδικα που παράγει ο decompiler.

1) Παράδειγμα απλής κλάσης main

Πηγαίος Κώδικας:

```
package client;

public class Main {

    public static void main(String[] args) {
        for (int i = 1; i < 1000; i++) {
            Runnable c = new Client("127.0.0.1", 12000);
            Thread t = new Thread(c);
            t.start();
        }
    }
}
```

Έξοδος bytecode viewer:

```
General Information:

Major Version: 50
Minor Version: 0
Superclass: java.lang.Object
Modifiers: 21
```

Fields:

Methods Information:

```
public <init> ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public static main    ([Ljava/lang/String;)V max stack: 4,   max locals: 4,   catch blocks:
0
```

Bytecode of main method:

```
0    iconst_1
1    istore_1
2    iload_1
3    sipush 1000
6    if_icmpge 41
9    new #2 = Class client.Client
12   dup
13   ldc #3 = "127.0.0.1"
15   sipush 12000
18   invokespecial #4 = Method client.Client.<init>((Ljava/lang/String;I)V)
21   astore_2
22   new #5 = Class java.lang.Thread
25   dup
26   aload_2
27   invokespecial #6 = Method java.lang.Thread.<init>((Ljava/lang/Runnable;)V)
30   astore_3
31   aload_3
32   invokevirtual #7 = Method java.lang.Thread.start(()V)
35   iinc 1 1
38   goto 2
41   return
```

Bytecode of Main() (constructor):

```
0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>(()V)
4    return
```

Εξοδος decompiler:

```
package client;

import client.Client;

public class Main{

    public Main(){
        super();
    }

    public static void main(String[ ] var0){
        int var1 = 1;
        while(var1 < 1000){
            Client var2 = new Client("127.0.0.1",12000);
            Thread var3 = new Thread(var2);
            var3.start();
            var1+= 1;
        }
    }
}
```

```
}
```

2) Παράδειγμα απλής κλάσης main

Πηγαίος Κώδικας:

```
public class Item {

    private static int uniqueId = 1;

    private int itemId;
    private String itemName;

    public Item(String itemName)
    {
        this.itemName = itemName;
        itemId = uniqueId;
        uniqueId++;
    }
}
```

Έξοδος bytecode viewer:

Methods Information:

```
public <init> (Ljava/lang/String;)V    max stack: 2,    max locals: 2,    catch blocks: 0
static <clinit> ()V                    max stack: 1,    max locals: 0,    catch blocks: 0
```

Bytecode of <clinit> method:

```
0    iconst_1
1    putstatic #3 = Field Item.uniqueId(I)
4    return
```

Bytecode of Item(java.lang.String) (constructor):

```
0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>(()V)
4    aload_0
5    aload_1
6    putfield #2 = Field Item.itemName(Ljava/lang/String;)
9    aload_0
10   getstatic #3 = Field Item.uniqueId(I)
13   putfield #4 = Field Item.itemId(I)
16   getstatic #3 = Field Item.uniqueId(I)
19   iconst_1
20   iadd
21   putstatic #3 = Field Item.uniqueId(I)
24   return
```

Έξοδος decompiler:

```
public class Item{
```

```

    private static int uniqueId;
    private int itemId;
    private String itemName;

    public Item(String var1){
        super();
        this.itemName = var1;
        this.itemId = uniqueId;
        uniqueId = (1+uniqueId);
    }

    static void <clinit>(){
        uniqueId = 1;
    }
}

```

3) Παράδειγμα βρόγχου while

Πηγαίος Κώδικας:

```

public class whilettest{
    public void spin(){
        int i = 0;
        int j = 0;
        while(i < 100){
            j=1;
            i+=1;
        }
        j = 3;
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 50
Minor Version: 0
Superclass: java.lang.Object
Modifiers: 21

```

Fields:

Methods Information:

```

public <init> ()V    max stack: 1,    max locals: 1,    catch blocks: 0
public spin      ()V    max stack: 2,    max locals: 3,    catch blocks: 0

```

Bytecode of spin method:

```

0    iconst_0
1    istore_1
2    iconst_0
3    istore_2
4    iload_1

```

```

5    bipush 100
7    if_icmpge 18
10   iconst_1
11   istore_2
12   iinc 1 1
15   goto 4
18   iconst_3
19   istore_2
20   return

```

Bytecode of whiletest() (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>(()V)
4    return

```

Έξοδος decompiler:

```

public class whiletest{

    public whiletest(){
        super();
    }

    public void spin(){
        int var1 = 0;
        int var2 = 0;
        while(var1 < 100){
            var2 = 1;
            var1+= 1;
        }
        var2 = 3;
    }

}

```

4) Παράδειγμα if-else

Πηγαίος Κώδικας:

```

public class test{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        if(s!=null){
            i= 1;
        }
        else{
            i = 0;
        }
        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

Major Version: 50
 Minor Version: 0
 Superclass: java.lang.Object
 Modifiers: 21

Fields:

Methods Information:

public <init> ()V max stack: 1, max locals: 1, **catch** blocks: 0
public add ()I max stack: 1, max locals: 4, **catch** blocks: 0

Bytecode of add method:

```
0   iconst_0
1   istore_1
2   aconst_null
3   astore_2
4   ldc #2 = "abc"
6   astore_3
7   aload_2
8   ifnull 16
11  iconst_1
12  istore_1
13  goto 18
16  iconst_0
17  istore_1
18  iconst_1
19  ireturn
```

Bytecode of test() (constructor):

```
0   aload_0
1   invokespecial #1 = Method java.lang.Object.<init>()V
4   return
```

Έξοδος decompiler:

```
public class test{

    public test(){
        super();
    }

    public int add(){
        int var1 = 0;
        Object var2 = null;
        String var3 = "abc";
        if(var2 != null){
            var1 = 1;
        }
        else{
            var1 = 0;
        }
    }
}
```

```

    }
    return 1;
}
}

```

5) Παράδειγμα if - else if - else

Πηγαίος Κώδικας:

```

public class test4{
    public int add(){
        int i=3;
        int j=4;
        if(i==0){
            j= 1;
        }
        else if(i==9){
            j = 0;
        }
        else{
            j=4;
        }
        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 50
Minor Version: 0
Superclass: java.lang.Object
Modifiers: 21

```

Fields:

Methods Information:

```

public <init> ()V    max stack: 1,    max locals: 1,    catch blocks: 0
public add      ()I    max stack: 2,    max locals: 3,    catch blocks: 0

```

Bytecode of add method:

```

0    iconst_3
1    istore_1
2    iconst_4
3    istore_2
4    iload_1
5    ifne 13
8    iconst_1
9    istore_2
10   goto 26
13   iload_1
14   bipush 9
16   if_icmpne 24

```

```

19    iconst_0
20    istore_2
21    goto 26
24    iconst_4
25    istore_2
26    iconst_1
27    ireturn

```

Bytecode of test4() (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    return

```

Έξοδος decompiler:

```

public class test4{

    public test4(){
        super();
    }

    public int add(){
        int var1 = 3;
        int var2 = 4;
        if(var1 == 0){
            var2 = 1;
        }
        else if(var1 == 9){
            var2 = 0;
        }
        else{
            var2 = 4;
        }
        return 1;
    }
}

```

6) Παράδειγμα βρόγχου do-while

Πηγαίος Κώδικας:

```

public class dowhile{
    public void spin() {
        int i = 0;
        do{
            int j=1;
            i+=1;
        }while(i < 100);
    }
}

```

Έξοδος bytecode viewer:

General Information:

```
Major Version: 50
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21
```

Fields:

Methods Information:

```
public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public spin     ()V      max stack: 2,   max locals: 3,   catch blocks: 0
```

Bytecode of spin method:

```
0   iconst_0
1   istore_1
2   iconst_1
3   istore_2
4   iinc 1 1
7   iload_1
8   bipush 100
10  if_icmplt 2
13  return
```

Bytecode of dowhile() (constructor):

```
0   aload_0
1   invokespecial #1 = Method java.lang.Object.<init>()V
4   return
```

Έξοδος decompiler:

```
public class dowhile{

    public dowhile(){
        super();
    }

    public void spin(){
        int var1 = 0;
        do{
            int var2 = 1;
            var1+= 1;
        }while(var1 < 100);
    }
}
```

7) Παράδειγμα βρόγχου for

Πηγαίος Κώδικας:

```
public class fortest{
    public void spin() {
```

```

        int i;
        int j;
        for (i = 0; i < 100; i+=1) {
            j=1;
        }
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 65
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Methods Information:

```

public <init> ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public spin    ()V      max stack: 2,   max locals: 3,   catch blocks: 0

```

Bytecode of spin method:

```

0   iconst_0
1   istore_1
2   iload_1
3   bipush 100
5   if_icmpge 16
8   iconst_1
9   istore_2
10  iinc 1 1
13  goto 2
16  return

```

Bytecode of forttest() (constructor):

```

0   aload_0
1   invokespecial #1 = Method java.lang.Object.<init>(()V)
4   return

```

Έξοδος decompiler:

```

public class forttest{

    public forttest(){
        super();
    }

    public void spin(){
        int var1 = 0;
        while(var1 < 100){
            int var2 = 1;
            var1+= 1;
        }
    }
}

```

```

    }
}
}

```

8) Παράδειγμα if - else με εμφωλευμένο if

Πηγαίος Κώδικας:

```

public class nestedifs22{

    public nestedifs22(){
        super();
    }

    public int add(){
        Object o = new Object();
        int i=0;
        String s=null;
        String r = "abc";
        if(s!=null){
            if(i==0){
                r = "av";
            }
        }
        else{
            i = 0;
        }
        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 65
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Methods Information:

```

public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add      ()I      max stack: 2,   max locals: 5,   catch blocks: 0

```

Bytecode of add method:

```

0    new #2 = Class java.lang.Object
3    dup
4    invokespecial #1 = Method java.lang.Object.<init>()V
7    astore_1
8    iconst_0
9    istore_2
10   aconst_null

```

```

11    astore_3
12    ldc #7 = "abc"
14    astore 4
16    aload_3
17    ifnull 31
20    iload_2
21    ifne 33
24    ldc #9 = "av"
26    astore 4
28    goto 33
31    iconst_0
32    istore_2
33    iconst_1
34    ireturn

```

Bytecode of nestedifs22() (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    return

```

Έξοδος decompiler:

```

public class nestedifs22{

    public nestedifs22(){
        super();
    }

    public int add(){
        Object var1 = new Object();
        int var2 = 0;
        Object var3 = null;
        String var4 = "abc";
        if(var3 != null){
            if(var2 == 0){
                var4 = "av";
            }
        }
        else{
            var2 = 0;
        }
        return 1;
    }
}

```

9) Παράδειγμα if - else με εμφωλευμένο if - else if - else

Πηγαίος Κώδικας:

```

public class nestedifs{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
    }
}

```

```

        if (s!=null){
            if (i==0){
                r = "av";
            }
            else if (i == 1){
                r = "ab";
            }
            else{
                r = "ac";
            }
            i= 1;
        }
        else{
            i = 0;
        }
        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 65
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Methods Information:

```

public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add      ()I      max stack: 2,   max locals: 4,   catch blocks: 0

```

Bytecode of add method:

```

0    iconst_0
1    istore_1
2    aconst_null
3    astore_2
4    ldc #7 = "abc"
6    astore_3
7    aload_2
8    ifnull 40
11   iload_1
12   ifne 21
15   ldc #9 = "av"
17   astore_3
18   goto 35
21   iload_1
22   iconst_1
23   if_icmpne 32
26   ldc #11 = "ab"
28   astore_3
29   goto 35
32   ldc #13 = "ac"
34   astore_3
35   iconst_1
36   istore_1
37   goto 42

```

```

40    iconst_0
41    istore_1
42    iconst_1
43    ireturn

```

Bytecode of nestedifs() (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>(()V)
4    return

```

Έξοδος decompiler:

```

public nestedifs(){
    super();
}

public int add(){
    int var1 = 0;
    Object var2 = null;
    String var3 = "abc";
    if(var2 != null){
        if(var1 == 0){
            var3 = "av";
        }
        else if(var1 == 1){
            var3 = "ab";
        }
        else{
            var3 = "ac";
        }
        var1 = 1;
    }
    else{
        var1 = 0;
    }
    return 1;
}
}

```

10) Παράδειγμα εμφωλευμένων while

Πηγαίος Κώδικας:

```

public class nestedwhiles{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        i=1;
        while(i<10){
            s = "abt";
            while(s!=null){
                i= 1;
            }
            i++;
        }
    }
}

```

```

        }
        i= 1;
        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 65
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Methods Information:

```

public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add      ()I      max stack: 2,   max locals: 4,   catch blocks: 0

```

Bytecode of add method:

```

0   iconst_0
1   istore_1
2   aconst_null
3   astore_2
4   ldc #7 = "abc"
6   astore_3
7   iconst_1
8   istore_1
9   iload_1
10  bipush 10
12  if_icmpge 33
15  ldc #9 = "abt"
17  astore_2
18  aload_2
19  ifnull 27
22  iconst_1
23  istore_1
24  goto 18
27  iinc 1 1
30  goto 9
33  iconst_1
34  istore_1
35  iconst_1
36  ireturn

```

Bytecode of nestedwhiles() (constructor):

```

0   aload_0
1   invokespecial #1 = Method java.lang.Object.<init>()V
4   return

```

Έξοδος decompiler:

```

public class nestedwhiles{

    public nestedwhiles(){
        super();
    }

    public int add(){
        int var1 = 0;
        String var2 = null;
        String var3 = "abc";
        var1 = 1;
        while(var1 < 10){
            var2 = "abt";
            while(var2 != null){
                var1 = 1;
            }
            var1+= 1;
        }
        var1 = 1;
        return 1;
    }
}

```

11) Παράδειγμα εμφωλευμένων do-while

Πηγαίος Κώδικας:

```

public class nesteddowhiles{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        i=1;
        do{
            s = "abt";
            do{
                i= 1;
            }while(s!=null);
            i++;
        }while(i<10);
        i= 1;
        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

Major Version: 65
 Minor Version: 0
 Superclass: java.lang.Object
 Modifiers: 21

Fields:

Methods Information:

```
public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add      ()I      max stack: 2,   max locals: 4,   catch blocks: 0
```

Bytecode of add method:

```
0    iconst_0
1    istore_1
2    aconst_null
3    astore_2
4    ldc #7 = "abc"
6    astore_3
7    iconst_1
8    istore_1
9    ldc #9 = "abt"
11   astore_2
12   iconst_1
13   istore_1
14   aload_2
15   ifnonnull 12
18   iinc 1 1
21   iload_1
22   bipush 10
24   if_icmplt 9
27   iconst_1
28   istore_1
29   iconst_1
30   ireturn
```

Bytecode of nesteddowhiles() (constructor):

```
0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    return
```

Έξοδος decompiler:

```
public class nesteddowhiles{

    public nesteddowhiles(){
        super();
    }

    public int add(){
        int var1 = 0;
        String var2 = null;
        String var3 = "abc";
        var1 = 1;
        do{
            var2 = "abt";
            do{
                var1 = 1;
            }while(var2 != null);
            var1+= 1;
        }while(var1 < 10);
        var1 = 1;
        return 1;
    }
}
```

```
}
```

12) Παράδειγμα do - while με εμφωλευμένο while

Πηγαίος Κώδικας:

```
public class dowhilewhile{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        i=1;
        do{
            s = "abt";
            while(s!=null){
                i= 1;
            }
            i++;
        }while(i<10);
        i= 1;
        return 1;
    }
}
```

Έξοδος bytecode viewer:

General Information:

```
Major Version: 65
Minor Version: 0
Superclass: java.lang.Object
Modifiers: 21
```

Fields:

Methods Information:

```
public <init> ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add      ()I      max stack: 2,   max locals: 4,   catch blocks: 0
```

Bytecode of add method:

```
0    iconst_0
1    istore_1
2    aconst_null
3    astore_2
4    ldc #7 = "abc"
6    astore_3
7    iconst_1
8    istore_1
9    ldc #9 = "abt"
11   astore_2
12   aload_2
13   ifnull 21
16   iconst_1
17   istore_1
```

```

18    goto 12
21    iinc 1 1
24    iload_1
25    bipush 10
27    if_icmplt 9
30    iconst_1
31    istore_1
32    iconst_1
33    ireturn

```

Bytecode of dowhilewhile() (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    return

```

Έξοδος decompiler:

```

public class dowhilewhile{

    public dowhilewhile(){
        super();
    }

    public int add(){
        int var1 = 0;
        String var2 = null;
        String var3 = "abc";
        var1 = 1;
        do{
            var2 = "abt";
            while(var2 != null){
                var1 = 1;
            }
            var1+= 1;
        }while(var1 < 10);
        var1 = 1;
        return 1;
    }
}

```

13) Παράδειγμα while με εμφωλευμένο do - while

Πηγαίος Κώδικας:

```

public class whiledowhile{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        i=1;
        while(i<10){
            s = "abt";
            do{
                i= 1;

```

```

        } while (s != null);
        i++;
    }
    i = 1;
    return 1;
}
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 65
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Methods Information:

```

public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add      ()I      max stack: 2,   max locals: 4,   catch blocks: 0

```

Bytecode of add method:

```

0   iconst_0
1   istore_1
2   aconst_null
3   astore_2
4   ldc #7 = "abc"
6   astore_3
7   iconst_1
8   istore_1
9   iload_1
10  bipush 10
12  if_icmpge 30
15  ldc #9 = "abt"
17  astore_2
18  iconst_1
19  istore_1
20  aload_2
21  ifnonnull 18
24  iinc 1 1
27  goto 9
30  iconst_1
31  istore_1
32  iconst_1
33  ireturn

```

Bytecode of whiledowhile() (constructor):

```

0   aload_0
1   invokespecial #1 = Method java.lang.Object.<init>()V
4   return

```

Έξοδος decompiler:

```

public class whiledowhile{

    public whiledowhile(){
        super();
    }

    public int add(){
        int var1 = 0;
        String var2 = null;
        String var3 = "abc";
        var1 = 1;
        while(var1 < 10){
            var2 = "abt";
            do{
                var1 = 1;
            }while(var2 != null);
            var1+= 1;
        }
        var1 = 1;
        return 1;
    }
}

```

14) Παράδειγμα while με εμφωλευμένο if - else

Πηγαίος Κώδικας:

```

public class whileif{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        i=1;
        while(i<10){
            if(s!=null){
                i= 1;
            }
            else{
                i = 0;
            }
            i++;
        }
        i= 1;
        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 65
Minor Version: 0
Superclass: java.lang.Object
Modifiers: 21

```

Fields:

Methods Information:

```
public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add      ()I      max stack: 2,   max locals: 4,   catch blocks: 0
```

Bytecode of add method:

```
0    iconst_0
1    istore_1
2    aconst_null
3    astore_2
4    ldc #7 = "abc"
6    astore_3
7    iconst_1
8    istore_1
9    iload_1
10   bipush 10
12   if_icmpge 32
15   aload_2
16   ifnull 24
19   iconst_1
20   istore_1
21   goto 26
24   iconst_0
25   istore_1
26   iinc 1 1
29   goto 9
32   iconst_1
33   istore_1
34   iconst_1
35   ireturn
```

Bytecode of whileif() (constructor):

```
0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>(()V)
4    return
```

Έξοδος decompiler:

```
public class whileif{

    public whileif(){
        super();
    }

    public int add(){
        int var1 = 0;
        Object var2 = null;
        String var3 = "abc";
        var1 = 1;
        while(var1 < 10){
            if(var2 != null){
                var1 = 1;
            }
            else{
```

```

        var1 = 0;
    }
    var1+= 1;
}
var1 = 1;
return 1;
}
}

```

15) Παράδειγμα do - while με εμφωλευμένο if - else

Πηγαίος Κώδικας:

```

public class dowhileif{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        i=1;
        do{
            if(s!=null){
                i = 1;
            }
            else{
                i = 0;
            }
            i++;
        }while(i<10);
        i= 1;
        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 65
Minor Version: 0
Superclass: java.lang.Object
Modifiers: 21

```

Fields:

Methods Information:

```

public <init> ()V    max stack: 1,    max locals: 1,    catch blocks: 0
public add      ()I    max stack: 2,    max locals: 4,    catch blocks: 0

```

Bytecode of add method:

```

0    iconst_0
1    istore_1
2    aconst_null
3    astore_2
4    ldc #7 = "abc"

```

```

6    astore_3
7    iconst_1
8    istore_1
9    aload_2
10   ifnull 18
13   iconst_1
14   istore_1
15   goto 20
18   iconst_0
19   istore_1
20   iinc 1 1
23   iload_1
24   bipush 10
26   if_icmplt 9
29   iconst_1
30   istore_1
31   iconst_1
32   ireturn

```

Bytecode of dowhileif() (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    return

```

Έξοδος decompiler:

```

public class dowhileif{

    public dowhileif(){
        super();
    }

    public int add(){
        int var1 = 0;
        Object var2 = null;
        String var3 = "abc";
        var1 = 1;
        do{
            if(var2 != null){
                var1 = 1;
            }
            else{
                var1 = 0;
            }
            var1+= 1;
        }while(var1 < 10);
        var1 = 1;
        return 1;
    }

}

```

16) Παράδειγμα if με εμφωλευμένο while

Πηγαίος Κώδικας:

```

public class ifwhile{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        if(s!=null){
            i++;
            while(i<10){
                i++;
            }
            i= 1;
        }
        else{
            i = 0;
        }
        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 65
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Methods Information:

```

public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add      ()I      max stack: 2,   max locals: 4,   catch blocks: 0

```

Bytecode of add method:

```

0    iconst_0
1    istore_1
2    aconst_null
3    astore_2
4    ldc #7 = "abc"
6    astore_3
7    aload_2
8    ifnull 31
11   iinc 1 1
14   iload_1
15   bipush 10
17   if_icmpge 26
20   iinc 1 1
23   goto 14
26   iconst_1
27   istore_1
28   goto 33
31   iconst_0
32   istore_1
33   iconst_1
34   ireturn

```

Bytecode of ifwhile() (constructor):

```
0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    return
```

Έξοδος decompiler:

```
public class ifwhile{

    public ifwhile(){
        super();
    }

    public int add(){
        int var1 = 0;
        Object var2 = null;
        String var3 = "abc";
        if(var2 != null){
            var1+= 1;
            while(var1 < 10){
                var1+= 1;
            }
            var1 = 1;
        }
        else{
            var1 = 0;
        }
        return 1;
    }
}
```

17) Παράδειγμα if με εμφωλευμένο do - while

Πηγαίος Κώδικας:

```
public class ifdowhile{
    public int add(){
        int i=0;
        String s=null;
        String r = "abc";
        if(s!=null){
            i=1;
            do{
                i++;
            }while(i<10);
            i= 1;
        }
        else{
            i = 0;
        }
        return 1;
    }
}
```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 65
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Methods Information:

```

public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add       ()I      max stack: 2,   max locals: 4,   catch blocks: 0

```

Bytecode of add method:

```

0    iconst_0
1    istore_1
2    aconst_null
3    astore_2
4    ldc #7 = "abc"
6    astore_3
7    aload_2
8    ifnull 27
11   iconst_1
12   istore_1
13   iinc 1 1
16   iload_1
17   bipush 10
19   if_icmplt 13
22   iconst_1
23   istore_1
24   goto 29
27   iconst_0
28   istore_1
29   iconst_1
30   ireturn

```

Bytecode of ifdowhile() (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    return

```

Έξοδος decompiler:

```

public class ifdowhile{

    public ifdowhile(){
        super();
    }

    public int add(){
        int var1 = 0;
        Object var2 = null;
        String var3 = "abc";
        if(var2 != null){

```

```

        var1 = 1;
        do{
            var1+= 1;
        }while(var1 < 10);
        var1 = 1;
    }
    else{
        var1 = 0;
    }
    return 1;
}
}

```

18) Παράδειγμα δήλωσης αριθμού τύπου double

Πηγαίος Κώδικας:

```

public class doubletest{
    public int add(){
        double d = 2.2;
        int i = 1000000;
        long l1 = 1;
        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

Major Version: 65
 Minor Version: 0
 Superclass: java.lang.Object
 Modifiers: 21

Fields:

Methods Information:

public <init>	()V	max stack: 1,	max locals: 1,	catch blocks: 0
public add	()I	max stack: 2,	max locals: 6,	catch blocks: 0

Bytecode of add method:

```

0    ldc2_w #7 = int 2.2
3    dstore_1
4    ldc #9 = int 1000000
6    istore_3
7    lconst_1
8    lstore 4
10   iconst_1
11   ireturn

```

Bytecode of doubletest() (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    return

```

Έξοδος decompiler:

```

public class doubletest{

    public doubletest(){
        super();
    }

    public int add(){
        double var1 = 2.2;
        int var3 = 1000000;
        long var4 = 1;
        return 1;
    }
}

```

19) Παράδειγμα πινάκων πολλαπλών διαστάσεων

Πηγαίος Κώδικας:

```

public class Hello2{
    public int add(float g,String s,int y, int x, String u){
        int [][][] a = new int [10][5][7];
        a[2][3][6] = 1;
        y = a.length;
        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 65
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Methods Information:

```

public <init> ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add      (FLjava/lang/String;IILjava/lang/String;)I  max stack: 3,   max locals: 7,
    catch blocks: 0

```

Bytecode of add method:

```

0    bipush 10
2    iconst_5
3    bipush 7
5    multianewarray #7 = Class [[[I
9    astore 6
11   aload 6
13   iconst_2
14   aaload
15   iconst_3
16   aaload
17   bipush 6
19   iconst_1
20   iastore
21   aload 6
23   arraylength
24   istore_3
25   iconst_1
26   ireturn

```

Bytecode of Hello2() (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>(()V)
4    return

```

Έξοδος decompiler:

```

public class Hello2{

    public Hello2(){
        super();
    }

    public int add(float var1, String var2, int var3, int var4, String var5){
        int[ ][ ][ ] var6 = new int[10][5][7];
        var6[2][3][6] = 1;
        var3 = var6.length;
        return 1;
    }
}

```

20) Παράδειγμα που περιέχει τη δομή switch

Πηγαίος Κώδικας:

```

package javassist.javassistextended;
public class Hello {

    protected int w;
    private int l;
    private int m;
    long ln;
    double d;
    float f;
    short s=4;
}

```

```

    byte b;
    boolean bool;
    char c;
    String str;
    int in = 1;
    String[] arr;
    int[] a;

    public Hello(int x){
        this.l = x;
    }

    private Hello(){
    }

    Hello(int x, byte g){
    }

    public void say() {
        int i=5;
        int j=10;
        int y=i+j;
    }

    public int calc(){
        return 5+3;
    }

    public int sw(int i){
        int j = 5;
        switch (i) {
            case 0: j=0;
                    break;
            case 1: j=1;
                    break;
            case 2: j=2;
                    break;
            default: j=-1;
                    break;
        }
        return j;
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 65
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Access Flag:	protected	Name:	w	Descriptor:	I
Access Flag:	private	Name:	l	Descriptor:	I
Access Flag:	private	Name:	m	Descriptor:	I
Access Flag:		Name:	ln	Descriptor:	J
Access Flag:		Name:	d	Descriptor:	D
Access Flag:		Name:	f	Descriptor:	F

Access Flag:	Name: s	Descriptor: S
Access Flag:	Name: b	Descriptor: B
Access Flag:	Name: bool	Descriptor: Z
Access Flag:	Name: c	Descriptor: C
Access Flag:	Name: str	Descriptor: Ljava/lang/String;
Access Flag:	Name: in	Descriptor: I
Access Flag:	Name: arr	Descriptor: [Ljava/lang/String;
Access Flag:	Name: a	Descriptor: [I

Methods Information:

```

public <init>    (I)V    max stack: 2,   max locals: 2,   catch blocks: 0
private <init>  ()V    max stack: 2,   max locals: 1,   catch blocks: 0
<init> (IB)V    max stack: 2,   max locals: 3,   catch blocks: 0
public say      ()V    max stack: 2,   max locals: 4,   catch blocks: 0
public calc     ()I    max stack: 1,   max locals: 1,   catch blocks: 0
public sw       (I)I    max stack: 1,   max locals: 3,   catch blocks: 0

```

Bytecode of say method:

```

0    iconst_5
1    istore_1
2    bipush 10
4    istore_2
5    iload_1
6    iload_2
7    iadd
8    istore_3
9    return

```

Bytecode of calc method:

```

0    bipush 8
2    ireturn

```

Bytecode of sw method:

```

0    iconst_5
1    istore_2
2    iload_1
3    tableswitch {
                        default: 43
                        0: 28
                        1: 33
                        2: 38}
28   iconst_0
29   istore_2
30   goto 45
33   iconst_1
34   istore_2
35   goto 45
38   iconst_2
39   istore_2
40   goto 45
43   iconst_m1
44   istore_2
45   iload_2
46   ireturn

```

Bytecode of Hello(int) (constructor):

```

0    aload_0

```

```

1    invokespecial #1 = Method java.lang.Object.<init>()V
4    aload_0
5    iconst_4
6    putfield #7 = Field javassist.javassistextended.Hello.s(S)
9    aload_0
10   iconst_1
11   putfield #13 = Field javassist.javassistextended.Hello.in(I)
14   aload_0
15   iload_1
16   putfield #17 = Field javassist.javassistextended.Hello.l(I)
19   return

```

Bytecode of Hello() (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    aload_0
5    iconst_4
6    putfield #7 = Field javassist.javassistextended.Hello.s(S)
9    aload_0
10   iconst_1
11   putfield #13 = Field javassist.javassistextended.Hello.in(I)
14   return

```

Bytecode of Hello(int,byte) (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    aload_0
5    iconst_4
6    putfield #7 = Field javassist.javassistextended.Hello.s(S)
9    aload_0
10   iconst_1
11   putfield #13 = Field javassist.javassistextended.Hello.in(I)
14   return

```

Έξοδος decompiler:

```
package javassist.javassistextended;
```

```

public class Hello{

    protected int w;
    private int l;
    private int m;
    long ln;
    double d;
    float f;
    short s;
    byte b;
    boolean bool;
    char c;
    String str;
    int in;
    String[ ] arr;
    int[ ] a;

    public Hello(int var1){
        super();
        this.s = 4;
        this.in = 1;
    }
}

```

```

        this.l = var1;
    }

    private Hello() {
        super();
        this.s = 4;
        this.in = 1;
    }

    Hello(int var1, byte var2) {
        super();
        this.s = 4;
        this.in = 1;
    }

    public void say() {
        int var1 = 5;
        int var2 = 10;
        int var3 = (var2+var1);
    }

    public int calc() {
        return 8;
    }

    public int sw(int var1) {
        int var2 = 5;
        switch(var1) {
            case 0: {
                var2 = 0;
                break;
            }
            case 1: {
                var2 = 1;
                break;
            }
            case 2: {
                var2 = 2;
                break;
            }
            default : {
                var2 = -1;
            }
        }
        return var2;
    }
}

```

21) Παράδειγμα try - catch - finally

Πηγαίος Κώδικας:

```

import java.io.*;

public class Main8 {

    private int integer;

    public static void main(String args) throws Exception {

```

```

        InputStream input = null;

        try{
            String t = null;
            args="test";
            t = args;
            input = new FileInputStream("c:\\data\\input-text.txt");
            int data = input.read();
            while(data != -1) {
                data = input.read();
            }
        }
        catch(IOException e){
            System.out.println("IO Exception caught");
        }
        catch(Exception e){
            System.out.println("Exception caught");
        }
        finally{
            if(input != null){
                input.close();
            }
        }
    }
}

```

Έξοδος bytecode viewer:

General Information:

Major Version: 50
 Minor Version: 0
 Superclass: java.lang.Object
 Modifiers: 21

Fields:

Access Flag: **private** Name: integer Descriptor: I

Methods Information:

public <init> ()V max stack: 1, max locals: 1, **catch** blocks: 0
public static main (Ljava/lang/String;)V max stack: 3, max locals: 5, **catch** blocks: 6

Bytecode of main method:

```

0    aconst_null
1    astore_1
2    aconst_null
3    astore_2
4    ldc #2 = "test"
6    astore_0
7    aload_0
8    astore_2
9    new #3 = Class java.io.FileInputStream
12   dup
13   ldc #4 = "c:\\data\\input-text.txt"
15   invokespecial #5 = Method java.io.FileInputStream.<init>((Ljava/lang/String;)V)
18   astore_1
19   aload_1
20   invokevirtual #6 = Method java.io.InputStream.read(())I)

```

```

23     istore_3
24     iload_3
25     iconst_m1
26     if_icmpeq 37
29     aload_1
30     invokevirtual #6 = Method java.io.InputStream.read(())I
33     istore_3
34     goto 24
37     aload_1
38     ifnull 101
41     aload_1
42     invokevirtual #7 = Method java.io.InputStream.close(())V
45     goto 101
48     astore_2
49     getstatic #9 = Field java.lang.System.out(Ljava/io/PrintStream;)
52     ldc #10 = "IO Exception caught"
54     invokevirtual #11 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
57     aload_1
58     ifnull 101
61     aload_1
62     invokevirtual #7 = Method java.io.InputStream.close(())V
65     goto 101
68     astore_2
69     getstatic #9 = Field java.lang.System.out(Ljava/io/PrintStream;)
72     ldc #13 = "Exception caught"
74     invokevirtual #11 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
77     aload_1
78     ifnull 101
81     aload_1
82     invokevirtual #7 = Method java.io.InputStream.close(())V
85     goto 101
88     astore 4
90     aload_1
91     ifnull 98
94     aload_1
95     invokevirtual #7 = Method java.io.InputStream.close(())V
98     aload 4
100    athrow
101    return

```

Bytecode of Main8() (constructor):

```

0     aload_0
1     invokespecial #1 = Method java.lang.Object.<init>()V
4     return

```

Έξοδος decompiler:

```

import java.io.PrintStream;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.InputStream;

public class Main8{

    private int integer;

    public Main8(){
        super();
    }

```

```

public static void main(String var0) throws Exception(){
    FileInputStream var1 = null;
    try{
        String var2 = null;
        var0 = "test";
        var2 = var0;
        var1 = new FileInputStream("c:\\data\\input-text.txt");
        int var3 = var1.read();
        while(var3 != -1){
            var3 = var1.read();
        }
    }
    catch(IOException e1){
        System.out.println("IO Exception caught");
    }
    catch(Exception e2){
        System.out.println("Exception caught");
    }
    finally{
        if(var1 != null){
            var1.close();
        }
    }
}
}

```

22) Παράδειγμα εμφωλευμένων try - catch - finally

Πηγαίος Κώδικας:

```

import java.io.*;

public class Main17{

    private int integer;

    public static void main(String args) throws Exception{

        InputStream input = null;

        try{
            String t = null;
            args="test";
            t = args;
            try {
                input = null;
            }
            catch (Exception e1) {
                System.out.println("Exception 1 caught");
            }
            finally{
                System.out.println("finally 1");
            }
            input = new FileInputStream("c:\\data\\input-text.txt");
            int data = input.read();
            while(data != -1) {
                data = input.read();
            }
        }
        catch(IOException e){

```

```

        try{
            input = null;
            System.out.println("try 2");
        }
        catch (Exception e1) {
            System.out.println("Exception 2 caught");
        }
        finally{
            System.out.println("finally 2");
        }
    }
    catch(Exception e2){
        System.out.println("Exception 3 caught");
    }
    finally{
        try{
            System.out.println("try 3");
            input = null;
        }
        catch (Exception e1) {
            System.out.println("Exception 4 caught");
        }
    }
}
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 50
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

```

Access Flag:    private Name:    integer Descriptor:    I

```

Methods Information:

```

public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public static  main    (Ljava/lang/String;)V  max stack: 3,   max locals: 8,   catch blocks:
18

```

Bytecode of main method:

```

0    aconst_null
1    astore_1
2    aconst_null
3    astore_2
4    ldc #2 = "test"
6    astore_0
7    aload_0
8    astore_2
9    aconst_null
10   astore_1
11   getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
14   ldc #4 = "finally 1"
16   invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
19   goto 55
22   astore_3

```

```

23     getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
26     ldc #7 = "Exception 1 caught"
28     invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
31     getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
34     ldc #4 = "finally 1"
36     invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
39     goto 55
42     astore 4
44     getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
47     ldc #4 = "finally 1"
49     invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
52     aload 4
54     throw
55     new #8 = Class java.io.FileInputStream
58     dup
59     ldc #9 = "c:\\data\\input-text.txt"
61     invokespecial #10 = Method java.io.FileInputStream.<init>((Ljava/lang/String;)V)
64     astore_1
65     aload_1
66     invokevirtual #11 = Method java.io.InputStream.read()I
69     istore_3
70     iload_3
71     iconst_m1
72     if_icmpeq 83
75     aload_1
76     invokevirtual #11 = Method java.io.InputStream.read()I
79     istore_3
80     goto 70
83     getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
86     ldc #12 = "try 3"
88     invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
91     aconst_null
92     astore_1
93     goto 250
96     astore_2
97     getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
100    ldc #13 = "Exception 4 caught"
102    invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
105    goto 250
108    astore_2
109    aconst_null
110    astore_1
111    getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
114    ldc #15 = "try 2"
116    invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
119    getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
122    ldc #16 = "finally 2"
124    invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
127    goto 163
130    astore_3
131    getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
134    ldc #17 = "Exception 2 caught"
136    invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
139    getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
142    ldc #16 = "finally 2"
144    invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
147    goto 163
150    astore 5
152    getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
155    ldc #16 = "finally 2"
157    invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
160    aload 5
162    throw
163    getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
166    ldc #12 = "try 3"

```

```

168     invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
171     aconst_null
172     astore_1
173     goto 250
176     astore_2
177     getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
180     ldc #13 = "Exception 4 caught"
182     invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
185     goto 250
188     astore_2
189     getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
192     ldc #18 = "Exception 3 caught"
194     invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
197     getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
200     ldc #12 = "try 3"
202     invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
205     aconst_null
206     astore_1
207     goto 250
210     astore_2
211     getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
214     ldc #13 = "Exception 4 caught"
216     invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
219     goto 250
222     astore_6
224     getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
227     ldc #12 = "try 3"
229     invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
232     aconst_null
233     astore_1
234     goto 247
237     astore_7
239     getstatic #3 = Field java.lang.System.out(Ljava/io/PrintStream;)
242     ldc #13 = "Exception 4 caught"
244     invokevirtual #5 = Method java.io.PrintStream.println((Ljava/lang/String;)V)
247     aload 6
249     athrow
250     return

```

Bytecode of Main17() (constructor):

```

0     aload_0
1     invokespecial #1 = Method java.lang.Object.<init>()V
4     return

```

Έξοδος decompiler:

```

import java.io.PrintStream;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.InputStream;

public class Main17{

    private int integer;

    public Main17(){
        super();
    }

    public static void main(String var0) throws Exception(){

```

```

FileInputStream var1 = null;
try{
    String var2 = null;
    var0 = "test";
    var2 = var0;
    try{
        var1 = null;
    }
    catch(Exception e1){
        System.out.println("Exception 1 caught");
    }
    finally{
        System.out.println("finally 1");
    }
    var1 = new FileInputStream("c:\\data\\input-text.txt");
    int var3 = var1.read();
    while(var3 != -1){
        var3 = var1.read();
    }
}
catch(IOException e2){
    try{
        var1 = null;
        System.out.println("try 2");
    }
    catch(Exception e3){
        System.out.println("Exception 2 caught");
    }
    finally{
        System.out.println("finally 2");
    }
}
catch(Exception e4){
    System.out.println("Exception 3 caught");
}
finally{
    try{
        System.out.println("try 3");
        var1 = null;
    }
    catch(Exception e5){
        System.out.println("Exception 4 caught");
    }
}
}

}

```

23) Παράδειγμα τελεστών ανάθεσης

Πηγαίος Κώδικας:

```

public class increment{
    public void spin() {
        int i = 0;
        int j;
        do{
            j=1;
            i-=1;
            i+=3;

```

```

        i+=128;
        i+=129;
        i+=32767;
        i+=32768;
        i-=32768;
        i-=129;
    }while( i < 100);
    do{
        i+=128;
    }while( i < 100);
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 50
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Methods Information:

```

public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public spin     ()V      max stack: 2,   max locals: 3,   catch blocks: 0

```

Bytecode of spin method:

```

0    iconst_0
1    istore_1
2    iconst_1
3    istore_2
4    iinc 1 255
7    iinc 1 3
10   wide
13   iinc 1 128
16   wide
19   iinc 1 129
22   wide
25   iinc 1 32767
28   iload_1
29   ldc #2 = int 32768
31   iadd
32   istore_1
33   wide
36   iinc 1 32768
39   wide
42   iinc 1 65407
45   iload_1
46   bipush 100
48   if_icmplt 2
51   wide
54   iinc 1 128
57   iload_1
58   bipush 100
60   if_icmplt 51
63   return

```

Bytecode of increment() (constructor):

```
0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    return
```

Έξοδος decompiler:

```
public class increment{

    public increment(){
        super();
    }

    public void spin(){
        int var1 = 0;
        do{
            int var2 = 1;
            var1-= 1;
            var1+= 3;
            var1+= 128;
            var1+= 129;
            var1+= 32767;
            var1 = (32768+var1);
            var1-= 32768;
            var1-= 129;
        }while(var1 < 100);
        do{
            var1+= 128;
        }while(var1 < 100);
    }

}
```

24) Παράδειγμα αρχικοποίησης πίνακα

Πηγαίος Κώδικας:

```
public class ArrayInit{
    public int add(int e, String y){
        int [] v = {2,3,4};
        int [][] w = {{1,2}, {3,4}};
        Integer [][] u = {{1,2}, {3,4}};
        String [][] o = {"a","b"}, {"c","d"};
        return 1;
    }
}
```

Έξοδος bytecode viewer:

General Information:

Major Version: 50

```

Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Methods Information:

```

public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add      (Ljava/lang/String;)I  max stack: 7,   max locals: 7,   catch blocks: 0

```

Bytecode of add method:

```

0    iconst_3
1    newarray int
3    dup
4    iconst_0
5    iconst_2
6    iastore
7    dup
8    iconst_1
9    iconst_3
10   iastore
11   dup
12   iconst_2
13   iconst_4
14   iastore
15   astore_3
16   iconst_2
17   anewarray #2 = Class [I
20   dup
21   iconst_0
22   iconst_2
23   newarray int
25   dup
26   iconst_0
27   iconst_1
28   iastore
29   dup
30   iconst_1
31   iconst_2
32   iastore
33   astore
34   dup
35   iconst_1
36   iconst_2
37   newarray int
39   dup
40   iconst_0
41   iconst_3
42   iastore
43   dup
44   iconst_1
45   iconst_4
46   iastore
47   astore
48   astore 4
50   iconst_2
51   anewarray #3 = Class [Ljava.lang.Integer;
54   dup
55   iconst_0
56   iconst_2
57   anewarray #4 = Class java.lang.Integer
60   dup

```

```

61     iconst_0
62     iconst_1
63     invokestatic #5 = Method java.lang.Integer.valueOf((I)Ljava/lang/Integer;)
66     astore
67     dup
68     iconst_1
69     iconst_2
70     invokestatic #5 = Method java.lang.Integer.valueOf((I)Ljava/lang/Integer;)
73     astore
74     astore
75     dup
76     iconst_1
77     iconst_2
78     anewarray #4 = Class java.lang.Integer
81     dup
82     iconst_0
83     iconst_3
84     invokestatic #5 = Method java.lang.Integer.valueOf((I)Ljava/lang/Integer;)
87     astore
88     dup
89     iconst_1
90     iconst_4
91     invokestatic #5 = Method java.lang.Integer.valueOf((I)Ljava/lang/Integer;)
94     astore
95     astore
96     astore 5
98     iconst_2
99     anewarray #6 = Class [Ljava.lang.String;
102    dup
103    iconst_0
104    iconst_2
105    anewarray #7 = Class java.lang.String
108    dup
109    iconst_0
110    ldc #8 = "a"
112    astore
113    dup
114    iconst_1
115    ldc #9 = "b"
117    astore
118    astore
119    dup
120    iconst_1
121    iconst_2
122    anewarray #7 = Class java.lang.String
125    dup
126    iconst_0
127    ldc #10 = "c"
129    astore
130    dup
131    iconst_1
132    ldc #11 = "d"
134    astore
135    astore
136    astore 6
138    iconst_1
139    ireturn

```

Bytecode of ArrayInit() (constructor):

```

0     aload_0
1     invokespecial #1 = Method java.lang.Object.<init>()V
4     return

```

Έξοδος decompiler:

```

public class ArrayInit{

    public ArrayInit(){
        super();
    }

    public int add(int var1, String var2){
        int[] var3 = new int[] {2,3,4};
        int[][] var4 = new int[][] {{1,2},{3,4}};
        Integer[][] var5 = new Integer[][] {{Integer.valueOf(1),Integer.valueOf(2)},{Integer.
            valueOf(3),Integer.valueOf(4)}};
        String[][] var6 = new String[][] {{{"a","b"},{"c","d"}}};
        return 1;
    }
}

```

25) Παράδειγμα Generics**Πηγαίος Κώδικας:**

```

import java.util.*;

public class generics{
    public int add(){
        HashMap<Integer, String> instructions = new LinkedHashMap<Integer, String>();
        Stack integers = new Stack();
        integers.push(new int[2][14]);
        integers.push(new int[] {1,2});
        return integers.size();
    }
}

```

Έξοδος bytecode viewer:

General Information:

Major Version: 50
 Minor Version: 0
 Superclass: java.lang.Object
 Modifiers: 21

Fields:

Methods Information:

public <init> ()V max stack: 1, max locals: 1, **catch** blocks: 0
public add ()I max stack: 5, max locals: 3, **catch** blocks: 0

Bytecode of add method:

0 new #2 = Class java.util.LinkedHashMap

```

3      dup
4      invokespecial #3 = Method java.util.LinkedHashMap.<init>(()V)
7      astore_1
8      new #4 = Class java.util.Stack
11     dup
12     invokespecial #5 = Method java.util.Stack.<init>(()V)
15     astore_2
16     aload_2
17     iconst_2
18     bipush 14
20     multianewarray #6 = Class [[I
24     invokevirtual #7 = Method java.util.Stack.push((Ljava/lang/Object;)Ljava/lang/Object;)
27     pop
28     aload_2
29     iconst_2
30     newarray int
32     dup
33     iconst_0
34     iconst_1
35     iastore
36     dup
37     iconst_1
38     iconst_2
39     iastore
40     invokevirtual #7 = Method java.util.Stack.push((Ljava/lang/Object;)Ljava/lang/Object;)
43     pop
44     aload_2
45     invokevirtual #8 = Method java.util.Stack.size()I
48     ireturn

```

Bytecode of generics() (constructor):

```

0      aload_0
1      invokespecial #1 = Method java.lang.Object.<init>(()V)
4      return

```

Έξοδος decompiler:

```

import java.util.LinkedHashMap;
import java.util.Stack;

public class generics{

    public generics(){
        super();
    }

    public int add(){
        LinkedHashMap var1 = new LinkedHashMap();
        Stack<Object> var2 = new Stack<Object>();
        var2.push(new int[2][14]);
        var2.push(new int[ ]{1,2});
        return var2.size();
    }
}

```

26) Παράδειγμα πολλαπλών συνθηκών if

Πηγαίος Κώδικας:

```

public class multipleifs5{
    public int add(){
        int i=0;
        String s=null;
        if(i!=3 && i!=4 || s!=null && i!=-1 && i!=2 || i!=5){
            i= 1;
        }
        if(i!=3 && i!=4 || i!=5 && i!=-1 && i!=2 || s!=null ){
            i= 1;
        }
        if(i!=3 && (i!=4 || i!=5) && i!=-1 && i!=2 || s!=null ){
            i= 1;
        }
        if((i!=3 || i!=4 && i!=7) && i!=5 && i!=-1 && i!=2 || s!=null ){
            i= 1;
        }
        if((i!=3 || i!=4 || i!=8) && i!=5 && i!=-1 && i!=2 || s!=null ){
            i= 1;
        }
        if((i!=4 || i!=5 || i!=6 && i!=7) && i!=-1 && i!=2 || s!=null ){
            i= 1;
        }
        if((i!=4 || i!=5 && i!=6 || i!=7) && i!=-1 && i!=2 || s!=null ){
            i= 1;
        }
        if((i!=7 && i!=4 || i!=5 || i!=6) && i!=-1 && i!=2 || s!=null ){
            i= 1;
        }
        if((i!=7 && i!=8 && i!=4 || i!=5 || i!=6) && i!=-1 && i!=2 || s!=null ){
            i= 1;
        }
        if((i!=7 && i!=8 || i!=4 && i!=5 || i!=6) && i!=-1 && i!=2 || s!=null ){
            i= 1;
        }
        if((i!=7 && i!=4) || (i!=5 || i!=6) && i!=-1 && i!=2 || s!=null ){
            i= 1;
        }
        if((i!=7 && i!=4) || (i!=5 || i!=6 && i!=-1 && i!=2) || s!=null ){
            i= 1;
        }
        if(i!=3 && (i!=4 || i!=5 || i!=6 && i!=7) && i!=-1 && i!=2 || s!=null ){
            i= 1;
        }
        if(i!=3 && (i!=7 && i!=4 || i!=5 || i!=6) && i!=-1 && i!=2 && s!=null ){
            i= 1;
        }
        if(i!=3 && (i!=7 && (i!=8 || i!=4) && (i!=5 || i!=6)) && i!=-1 && i!=2 && s!=
            null ){
            i= 1;
        }
        if(i!=3 && (i!=7 || i!=8 && (i!=4 || i!=5 || i!=6)) && i!=-1 && i!=2 && s!=null ){
            i= 1;
        }
        if(i!=3 || ((i!=7 || i!=8) && (i!=4 || i!=5 || i!=6)) && i!=-1 && i!=2 && s!=null
        ){
            i= 1;
        }
        if(i!=3 || ((i!=7 || i!=8) && (i!=4 || i!=5 || i!=6)) && i!=-1 && (i!=10 && i!=2
        || s!=null && i!=9)){
            i= 1;
        }
    }
}

```

```

        if (i != -1 && i != 2 || s != null && (i != 7 && i != 4 || i != 5 || i != 6)) {
            i = 1;
        }
        if ((i != 10 && i != 2 || s != null && i != 9) && i != 3 || ((i != 7 || i != 8) && (i != 4 || i != 5
            || i != 6)) && i != -1) {
            i = 1;
        }
        if (i != 11 && (i != 10 && i != 2 || s != null && i != 9) && i != 3 || ((i != 7 || i != 8) && (i
            != 4 || i != 5 || i != 6)) && i != -1) {
            i = 1;
        }

        return 1;
    }
}

```

Έξοδος bytecode viewer:

General Information:

```

Major Version: 50
Minor Version: 0
Superclass:    java.lang.Object
Modifiers:     21

```

Fields:

Methods Information:

```

public <init>    ()V      max stack: 1,   max locals: 1,   catch blocks: 0
public add      ()I      max stack: 2,   max locals: 3,   catch blocks: 0

```

Bytecode of add method:

```

0    iconst_0
1    istore_1
2    aconst_null
3    astore_2
4    iload_1
5    iconst_3
6    if_icmpeq 14
9    iload_1
10   iconst_4
11   if_icmpne 33
14   aload_2
15   ifnull 28
18   iload_1
19   iconst_m1
20   if_icmpeq 28
23   iload_1
24   iconst_2
25   if_icmpne 33
28   iload_1
29   iconst_5
30   if_icmpeq 35
33   iconst_1
34   istore_1
35   iload_1
36   iconst_3
37   if_icmpeq 45
40   iload_1

```

```
41     iconst_4
42     if_icmpne 64
45     iload_1
46     iconst_5
47     if_icmpeq 60
50     iload_1
51     iconst_m1
52     if_icmpeq 60
55     iload_1
56     iconst_2
57     if_icmpne 64
60     aload_2
61     ifnull 66
64     iconst_1
65     istore_1
66     iload_1
67     iconst_3
68     if_icmpeq 91
71     iload_1
72     iconst_4
73     if_icmpne 81
76     iload_1
77     iconst_5
78     if_icmpeq 91
81     iload_1
82     iconst_m1
83     if_icmpeq 91
86     iload_1
87     iconst_2
88     if_icmpne 95
91     aload_2
92     ifnull 97
95     iconst_1
96     istore_1
97     iload_1
98     iconst_3
99     if_icmpne 113
102    iload_1
103    iconst_4
104    if_icmpeq 128
107    iload_1
108    bipush 7
110    if_icmpeq 128
113    iload_1
114    iconst_5
115    if_icmpeq 128
118    iload_1
119    iconst_m1
120    if_icmpeq 128
123    iload_1
124    iconst_2
125    if_icmpne 132
128    aload_2
129    ifnull 134
132    iconst_1
133    istore_1
134    iload_1
135    iconst_3
136    if_icmpne 150
139    iload_1
140    iconst_4
141    if_icmpne 150
144    iload_1
145    bipush 8
147    if_icmpeq 165
```

```

150    iload_1
151    iconst_5
152    if_icmpeq 165
155    iload_1
156    iconst_m1
157    if_icmpeq 165
160    iload_1
161    iconst_2
162    if_icmpne 169
165    aload_2
166    ifnull 171
169    iconst_1
170    istore_1
171    iload_1
172    iconst_4
173    if_icmpne 193
176    iload_1
177    iconst_5
178    if_icmpne 193
181    iload_1
182    bipush 6
184    if_icmpeq 203
187    iload_1
188    bipush 7
190    if_icmpeq 203
193    iload_1
194    iconst_m1
195    if_icmpeq 203
198    iload_1
199    iconst_2
200    if_icmpne 207
203    aload_2
204    ifnull 209
207    iconst_1
208    istore_1
209    iload_1
210    iconst_4
211    if_icmpne 231
214    iload_1
215    iconst_5
216    if_icmpeq 225
219    iload_1
220    bipush 6
222    if_icmpne 231
225    iload_1
226    bipush 7
228    if_icmpeq 241
231    iload_1
232    iconst_m1
233    if_icmpeq 241
236    iload_1
237    iconst_2
238    if_icmpne 245
241    aload_2
242    ifnull 247
245    iconst_1
246    istore_1
247    iload_1
248    bipush 7
250    if_icmpeq 258
253    iload_1
254    iconst_4
255    if_icmpne 269
258    iload_1
259    iconst_5

```

```

260     if_icmpne 269
263     iload_1
264     bipush 6
266     if_icmpeq 279
269     iload_1
270     iconst_m1
271     if_icmpeq 279
274     iload_1
275     iconst_2
276     if_icmpne 283
279     aload_2
280     ifnull 285
283     iconst_1
284     istore_1
285     iload_1
286     bipush 7
288     if_icmpeq 302
291     iload_1
292     bipush 8
294     if_icmpeq 302
297     iload_1
298     iconst_4
299     if_icmpne 313
302     iload_1
303     iconst_5
304     if_icmpne 313
307     iload_1
308     bipush 6
310     if_icmpeq 323
313     iload_1
314     iconst_m1
315     if_icmpeq 323
318     iload_1
319     iconst_2
320     if_icmpne 327
323     aload_2
324     ifnull 329
327     iconst_1
328     istore_1
329     iload_1
330     bipush 7
332     if_icmpeq 341
335     iload_1
336     bipush 8
338     if_icmpne 357
341     iload_1
342     iconst_4
343     if_icmpeq 351
346     iload_1
347     iconst_5
348     if_icmpne 357
351     iload_1
352     bipush 6
354     if_icmpeq 367
357     iload_1
358     iconst_m1
359     if_icmpeq 367
362     iload_1
363     iconst_2
364     if_icmpne 371
367     aload_2
368     ifnull 373
371     iconst_1
372     istore_1
373     iload_1

```

```

374     bipush 7
376     if_icmpeq 384
379     iload_1
380     iconst_4
381     if_icmpne 409
384     iload_1
385     iconst_5
386     if_icmpne 395
389     iload_1
390     bipush 6
392     if_icmpeq 405
395     iload_1
396     iconst_m1
397     if_icmpeq 405
400     iload_1
401     iconst_2
402     if_icmpne 409
405     aload_2
406     ifnull 411
409     iconst_1
410     istore_1
411     iload_1
412     bipush 7
414     if_icmpeq 422
417     iload_1
418     iconst_4
419     if_icmpne 447
422     iload_1
423     iconst_5
424     if_icmpne 447
427     iload_1
428     bipush 6
430     if_icmpeq 443
433     iload_1
434     iconst_m1
435     if_icmpeq 443
438     iload_1
439     iconst_2
440     if_icmpne 447
443     aload_2
444     ifnull 449
447     iconst_1
448     istore_1
449     iload_1
450     iconst_3
451     if_icmpeq 486
454     iload_1
455     iconst_4
456     if_icmpne 476
459     iload_1
460     iconst_5
461     if_icmpne 476
464     iload_1
465     bipush 6
467     if_icmpeq 486
470     iload_1
471     bipush 7
473     if_icmpeq 486
476     iload_1
477     iconst_m1
478     if_icmpeq 486
481     iload_1
482     iconst_2
483     if_icmpne 490
486     aload_2

```

```

487     ifnull 492
490     iconst_1
491     istore_1
492     iload_1
493     iconst_3
494     if_icmpeq 535
497     iload_1
498     bipush 7
500     if_icmpeq 508
503     iload_1
504     iconst_4
505     if_icmpne 519
508     iload_1
509     iconst_5
510     if_icmpne 519
513     iload_1
514     bipush 6
516     if_icmpeq 535
519     iload_1
520     iconst_m1
521     if_icmpeq 535
524     iload_1
525     iconst_2
526     if_icmpeq 535
529     aload_2
530     ifnull 535
533     iconst_1
534     istore_1
535     iload_1
536     iconst_3
537     if_icmpeq 584
540     iload_1
541     bipush 7
543     if_icmpeq 584
546     iload_1
547     bipush 8
549     if_icmpne 557
552     iload_1
553     iconst_4
554     if_icmpeq 584
557     iload_1
558     iconst_5
559     if_icmpne 568
562     iload_1
563     bipush 6
565     if_icmpeq 584
568     iload_1
569     iconst_m1
570     if_icmpeq 584
573     iload_1
574     iconst_2
575     if_icmpeq 584
578     aload_2
579     ifnull 584
582     iconst_1
583     istore_1
584     iload_1
585     iconst_3
586     if_icmpeq 633
589     iload_1
590     bipush 7
592     if_icmpne 617
595     iload_1
596     bipush 8
598     if_icmpeq 633

```

```
601    iload_1
602    iconst_4
603    if_icmpne 617
606    iload_1
607    iconst_5
608    if_icmpne 617
611    iload_1
612    bipush 6
614    if_icmpeq 633
617    iload_1
618    iconst_m1
619    if_icmpeq 633
622    iload_1
623    iconst_2
624    if_icmpeq 633
627    aload_2
628    ifnull 633
631    iconst_1
632    istore_1
633    iload_1
634    iconst_3
635    if_icmpne 680
638    iload_1
639    bipush 7
641    if_icmpne 650
644    iload_1
645    bipush 8
647    if_icmpeq 682
650    iload_1
651    iconst_4
652    if_icmpne 666
655    iload_1
656    iconst_5
657    if_icmpne 666
660    iload_1
661    bipush 6
663    if_icmpeq 682
666    iload_1
667    iconst_m1
668    if_icmpeq 682
671    iload_1
672    iconst_2
673    if_icmpeq 682
676    aload_2
677    ifnull 682
680    iconst_1
681    istore_1
682    iload_1
683    iconst_3
684    if_icmpne 741
687    iload_1
688    bipush 7
690    if_icmpne 699
693    iload_1
694    bipush 8
696    if_icmpeq 743
699    iload_1
700    iconst_4
701    if_icmpne 715
704    iload_1
705    iconst_5
706    if_icmpne 715
709    iload_1
710    bipush 6
712    if_icmpeq 743
```

```
715      iload_1
716      iconst_m1
717      if_icmpeq 743
720      iload_1
721      bipush 10
723      if_icmpeq 731
726      iload_1
727      iconst_2
728      if_icmpne 741
731      aload_2
732      ifnull 743
735      iload_1
736      bipush 9
738      if_icmpeq 743
741      iconst_1
742      istore_1
743      iload_1
744      iconst_m1
745      if_icmpeq 753
748      iload_1
749      iconst_2
750      if_icmpne 779
753      aload_2
754      ifnull 781
757      iload_1
758      bipush 7
760      if_icmpeq 768
763      iload_1
764      iconst_4
765      if_icmpne 779
768      iload_1
769      iconst_5
770      if_icmpne 779
773      iload_1
774      bipush 6
776      if_icmpeq 781
779      iconst_1
780      istore_1
781      iload_1
782      bipush 10
784      if_icmpeq 792
787      iload_1
788      iconst_2
789      if_icmpne 802
792      aload_2
793      ifnull 807
796      iload_1
797      bipush 9
799      if_icmpeq 807
802      iload_1
803      iconst_3
804      if_icmpne 840
807      iload_1
808      bipush 7
810      if_icmpne 819
813      iload_1
814      bipush 8
816      if_icmpeq 842
819      iload_1
820      iconst_4
821      if_icmpne 835
824      iload_1
825      iconst_5
826      if_icmpne 835
829      iload_1
```

```

830    bipush 6
832    if_icmpeq 842
835    iload_1
836    iconst_m1
837    if_icmpeq 842
840    iconst_1
841    istore_1
842    iload_1
843    bipush 11
845    if_icmpeq 874
848    iload_1
849    bipush 10
851    if_icmpeq 859
854    iload_1
855    iconst_2
856    if_icmpne 869
859    aload_2
860    ifnull 874
863    iload_1
864    bipush 9
866    if_icmpeq 874
869    iload_1
870    iconst_3
871    if_icmpne 907
874    iload_1
875    bipush 7
877    if_icmpne 886
880    iload_1
881    bipush 8
883    if_icmpeq 909
886    iload_1
887    iconst_4
888    if_icmpne 902
891    iload_1
892    iconst_5
893    if_icmpne 902
896    iload_1
897    bipush 6
899    if_icmpeq 909
902    iload_1
903    iconst_m1
904    if_icmpeq 909
907    iconst_1
908    istore_1
909    iconst_1
910    ireturn

```

Bytecode of `multipleifs5()` (constructor):

```

0    aload_0
1    invokespecial #1 = Method java.lang.Object.<init>()V
4    return

```

Έξοδος decompiler:

```

public class multipleifs5{

    public multipleifs5(){
        super();
    }
}

```

```

public int add(){
    int var1 = 0;
    Object var2 = null;
    if(var1 != 3 && var1 != 4 || var2 != null && var1 != -1 && var1 != 2 || var1 != 5){
        var1 = 1;
    }
    if(var1 != 3 && var1 != 4 || var1 != 5 && var1 != -1 && var1 != 2 || var2 != null){
        var1 = 1;
    }
    if(var1 != 3 && (var1 != 4 || var1 != 5) && var1 != -1 && var1 != 2 || var2 != null){
        var1 = 1;
    }
    if((var1 != 3 || var1 != 4 && var1 != 7) && var1 != 5 && var1 != -1 && var1 != 2 || var2
        != null){
        var1 = 1;
    }
    if((var1 != 3 || var1 != 4 || var1 != 8) && var1 != 5 && var1 != -1 && var1 != 2 || var2
        != null){
        var1 = 1;
    }
    if((var1 != 4 || var1 != 5 || var1 != 6 && var1 != 7) && var1 != -1 && var1 != 2 || var2
        != null){
        var1 = 1;
    }
    if((var1 != 4 || var1 != 5 && var1 != 6 || var1 != 7) && var1 != -1 && var1 != 2 || var2
        != null){
        var1 = 1;
    }
    if((var1 != 4 || var1 != 5 || var1 != 6) && var1 != -1 && var1 != 2 || var2
        != null){
        var1 = 1;
    }
    if((var1 != 7 && var1 != 8 && var1 != 4 || var1 != 5 || var1 != 6) && var1 != -1 && var1
        != 2 || var2 != null){
        var1 = 1;
    }
    if((var1 != 7 && var1 != 8 || var1 != 4 && var1 != 5 || var1 != 6) && var1 != -1 && var1
        != 2 || var2 != null){
        var1 = 1;
    }
    if(var1 != 7 && var1 != 4 || (var1 != 5 || var1 != 6) && var1 != -1 && var1 != 2 || var2
        != null){
        var1 = 1;
    }
    if(var1 != 7 && var1 != 4 || var1 != 5 || var1 != 6 && var1 != -1 && var1 != 2 || var2 !=
        null){
        var1 = 1;
    }
    if(var1 != 3 && (var1 != 4 || var1 != 5 || var1 != 6 && var1 != 7) && var1 != -1 && var1
        != 2 || var2 != null){
        var1 = 1;
    }
    if(var1 != 3 && (var1 != 7 && var1 != 4 || var1 != 5 || var1 != 6) && var1 != -1 && var1
        != 2 && var2 != null){
        var1 = 1;
    }
    if(var1 != 3 && var1 != 7 && (var1 != 8 || var1 != 4) && (var1 != 5 || var1 != 6) && var1
        != -1 && var1 != 2 && var2 != null){
        var1 = 1;
    }
    if(var1 != 3 && (var1 != 7 || var1 != 8 && (var1 != 4 || var1 != 5 || var1 != 6)) && var1
        != -1 && var1 != 2 && var2 != null){
        var1 = 1;
    }
}

```

```

    if (var1 != 3 || (var1 != 7 || var1 != 8) && (var1 != 4 || var1 != 5 || var1 != 6) && var1
        != -1 && var1 != 2 && var2 != null){
        var1 = 1;
    }
    if (var1 != 3 || (var1 != 7 || var1 != 8) && (var1 != 4 || var1 != 5 || var1 != 6) && var1
        != -1 && (var1 != 10 && var1 != 2 || var2 != null && var1 != 9)){
        var1 = 1;
    }
    if (var1 != -1 && var1 != 2 || var2 != null && (var1 != 7 && var1 != 4 || var1 != 5 || var1
        != 6)){
        var1 = 1;
    }
    if ((var1 != 10 && var1 != 2 || var2 != null && var1 != 9) && var1 != 3 || (var1 != 7 ||
        var1 != 8) && (var1 != 4 || var1 != 5 || var1 != 6) && var1 != -1){
        var1 = 1;
    }
    if (var1 != 11 && (var1 != 10 && var1 != 2 || var2 != null && var1 != 9) && var1 != 3 || (
        var1 != 7 || var1 != 8) && (var1 != 4 || var1 != 5 || var1 != 6) && var1 != -1){
        var1 = 1;
    }
    return 1;
}
}

```

Κεφάλαιο Ζ΄

ΕΓΧΕΙΡΙΔΙΟ ΧΡΗΣΗΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

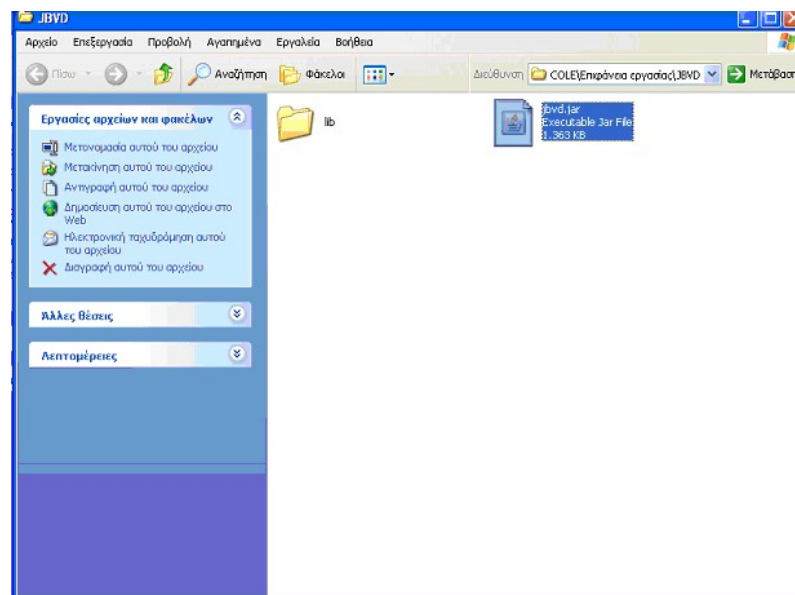
Ζ'.1 Περιγραφή της εφαρμογής

Ο χρήστης μέσω της εφαρμογής JBVD έχει 2 δυνατότητες: 1) να βλέπει τα bytecodes μιας κλάσης γραμμένης σε γλώσσα Java την οποία πρέπει να δώσει ως είσοδο στο πρόγραμμα (.java αρχείο). Η κλάση αυτή δεν πρέπει να έχει περάσει από φάση μεταγλώττισης αφού το ίδιο το πρόγραμμα αναλαμβάνει τη μεταγλώττιση της, 2) να εμφανίζει τον πηγαίο κώδικα αλλά και τα bytecodes μιας κλάσης παίρνοντας ως είσοδο ένα .class αρχείο.

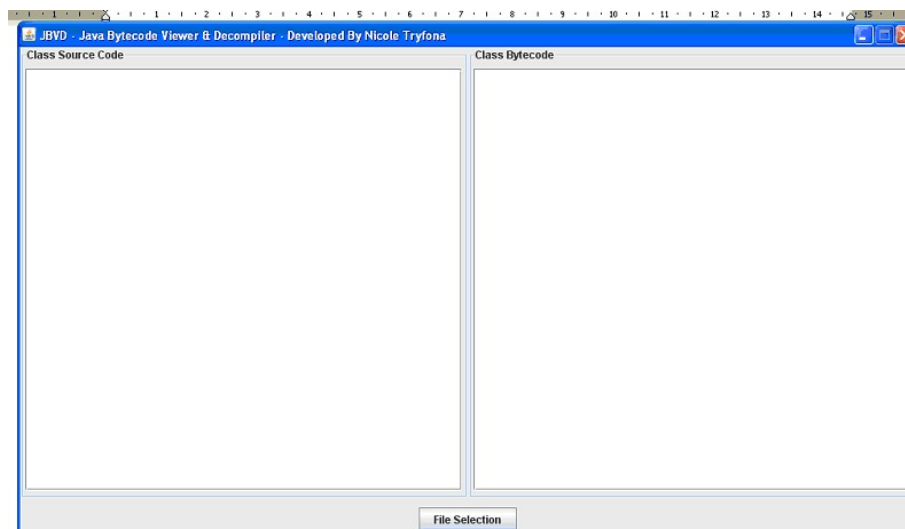
Ζ'.2 Εγκατάσταση και χρήση της εφαρμογής

Το πρόγραμμα JBVD είναι γραμμένο σε Java οπότε απαιτείται ο χρήστης να έχει εγκατεστημένο το java jdk και επιπλέον η έκδοση του jdk η οποία θα είναι εγκατεστημένη πρέπει να είναι τουλάχιστον η 1.6. Όσον αφορά την χρήση της εφαρμογής αυτή δεν απαιτεί καμία ιδιαίτερη εγκατάσταση. Συγκεκριμένα πρέπει να ακολουθηθούν τα εξής απλά βήματα:

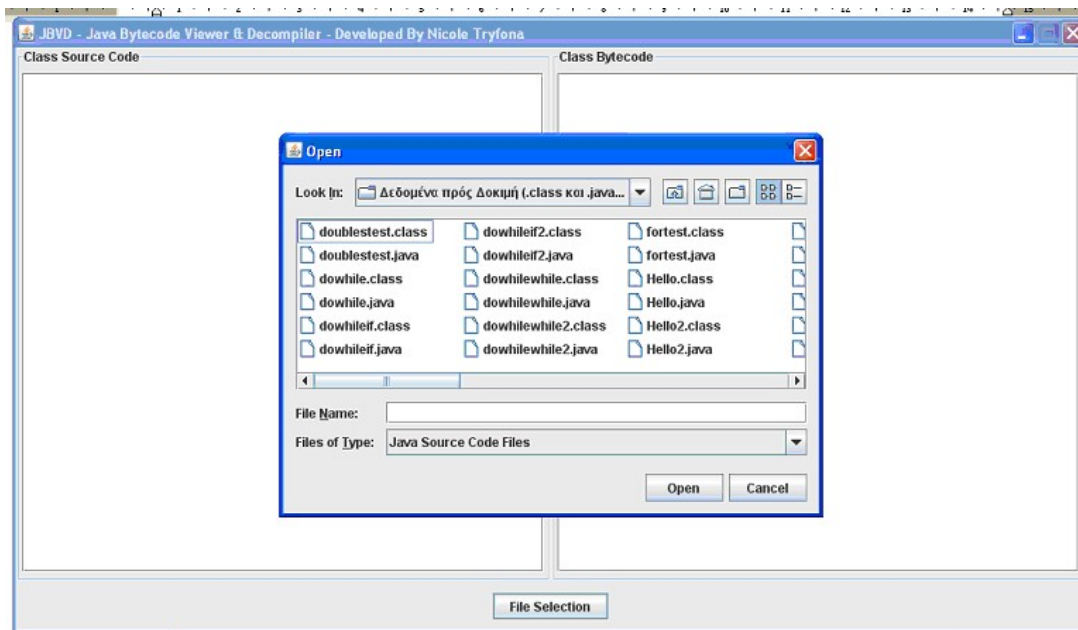
1. Το πρώτο βήμα είναι να αποσυμπίεστεί το αρχείο της εφαρμογής. 2. Αφού γίνει η αποσυμπίεση του αρχείου ο χρήστης πρέπει να ανοίξει τον φάκελο που περιέχει την εφαρμογή. Τα περιεχόμενα του φακέλου είναι τα ακόλουθα:



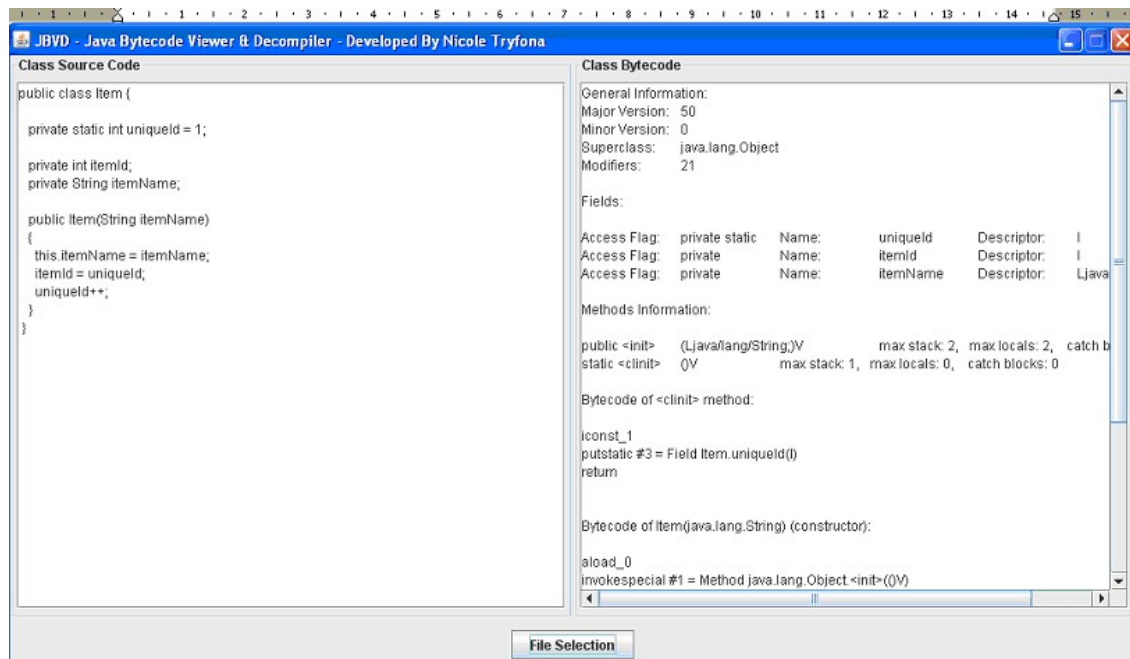
3. Το επόμενο βήμα είναι να γίνει διπλό κλικ στο αρχείο `jbvd.jar` όπου είναι και το εκτελέσιμο αρχείο της εφαρμογής. Ο φάκελος `lib` απεναντίας περιέχει τη βιβλιοθήκη `tools` η οποία είναι απαραίτητη για τη λειτουργία της εφαρμογής και συγκεκριμένα για να γίνει η μεταγλώττιση της κλάσης που θα δώσει ως είσοδο ο χρήστης. 4. Αφού ο χρήστης κάνει διπλό κλικ στο αρχείο `jbvd.jar` θα εμφανιστεί το ακόλουθο γραφικό περιβάλλον:



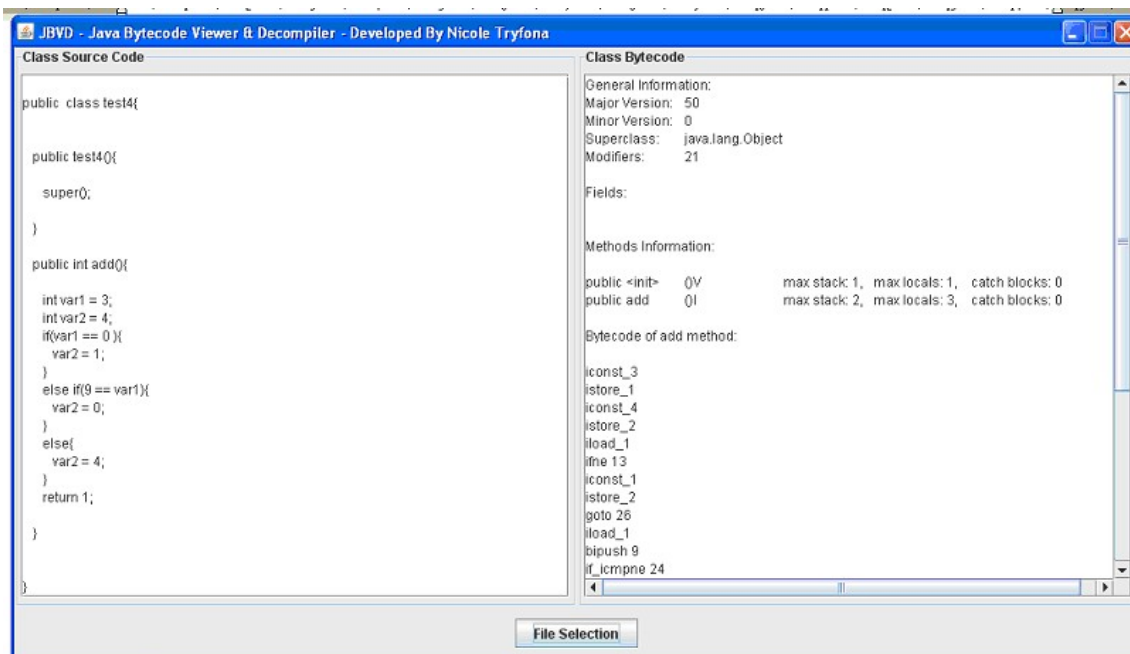
5. Το επόμενο βήμα είναι ο χρήστης να διαλέξει ένα `.java` ή ένα `.class` αρχείο. Αυτό γίνεται μέσω του κουμπιού `File Selection` το οποίο εμφανίζει ένα παράθυρο επιλογής αρχείου:



6. Στον χρήστη εμφανίζονται μόνον φάκελοι και αρχεία .java ή .class. Ο χρήστης διαλέγει το αρχείο που επιθυμεί και πατά Open. Έπειτα, αν έχει διαλέξει αρχείο java και δεν εμφανιστούν συντακτικά λάθη κατά τη φάση της μεταγλώττισης, στο αριστερό μέρος του κεντρικού παραθύρου της εφαρμογής (Class Source Code) θα εμφανιστεί ο πηγαίος κώδικας της κλάσης που επιλέχθηκε και στο δεξί μέρος (Class Bytecode) θα εμφανιστούν τα bytecodes της κλάσης:



Αν από την άλλη, έχει επιλέξει αρχείο class, τότε στο δεξί μέρος του κεντρικού παραθύρου της εφαρμογής (Class Bytecode) θα εμφανιστούν τα bytecodes της κλάσης ενώ στο δεξί (Class Source Code) θα εμφανιστεί ο πηγαίος κώδικας της κλάσης που επιλέχθηκε:



Ζ'.3 Δεδομένα εξόδου του προγράμματος

Οι πληροφορίες που εμφανίζονται στο δεξί μέρος του κεντρικού παραθύρου της εφαρμογής είναι οι εξής:

- Οι γενικές πληροφορίες της κλάσης (General Information)
- Τα πεδία (fields) της κλάσης
- Οι διεπαφές που υλοποιεί η κλάση (Interfaces)
- Οι γενικές πληροφορίες των μεθόδων (Methods Information)
- Τα bytecodes κάθε μεθόδου
- Όλα τα προαναφερθέντα για κάθε εμφωλευμένη κλάση.

Οι πληροφορίες που εμφανίζονται στο αριστερό μέρος του κεντρικού παραθύρου της εφαρμογής αφορούν τον πηγαίο κώδικα της κλάσης (περιέχει τα πεδία, τους constructors και όλες τις μεθόδους της).

Παράρτημα Α΄ : Πηγαίος Κώδικας

```
1 package jbv;
3 import javax.swing.text.*;
import java.util.*;
5
public class AbstractSyntaxTree {
7
    private static AbstractSyntaxTreeNode root;
9
    public AbstractSyntaxTree(int pcValue, String data){
11        root = new AbstractSyntaxTreeNode(pcValue, data);
    }
13
    public AbstractSyntaxTreeNode getRoot(){
15        return root;
    }
17
    public void setRoot(AbstractSyntaxTreeNode node){
19        root = node;
    }
21
    public void addNode(Stack <AbstractSyntaxTreeNode[]> branchNodeStack,
        AbstractSyntaxTreeNode currentNode, AbstractSyntaxTreeNode newNode, int
        currentPC){
23        AbstractSyntaxTreeNode previousNode;
        int leftPC = -1;
25        int rightPC = -1;
        int newPC = newNode.getPcValue();
27        do{
            previousNode = currentNode;
29            if(currentNode.getLeft() == null){
                leftPC = -1;
31            }
            else{
33                leftPC = currentNode.getLeft().getPcValue();
```

```

35         }
36         if(currentNode.getRight() == null){
37             rightPC = -1;
38         }
39         else{
40             rightPC = currentNode.getRight().getPcValue();
41         }
42         if((newPC <= rightPC && newPC > leftPC) || (rightPC == leftPC)){
43             currentNode = currentNode.getRight();
44         }
45         else if(newPC < rightPC && newPC < leftPC){
46             if(rightPC < leftPC){
47                 currentNode = currentNode.getRight();
48             }
49             else{
50                 currentNode = currentNode.getLeft();
51             }
52         }
53         else if(newPC > rightPC && newPC <= leftPC){
54             currentNode = currentNode.getLeft();
55         }
56         else if(newPC > rightPC && newPC > leftPC){
57             if(((leftPC > currentNode.getPcValue()) && rightPC < leftPC) ||
58                 currentNode.getData().startsWith("while")){
59                 currentNode = currentNode.getLeft();
60             }
61             else if(rightPC != -1 && rightPC < currentNode.getPcValue()){ //do
62                 while case
63                 currentNode = currentNode.getLeft();
64             }
65             else{
66                 currentNode = currentNode.getRight();
67             }
68         }
69     } while(currentNode != null && newPC >= currentNode.getPcValue());
70     if(previousNode.getRight() == currentNode){
71         previousNode.setRight(newNode);
72         while(previousNode.getLeft() != null && previousNode.getLeft().getPcValue()
73             < previousNode.getPcValue()){
74             AbstractSyntaxTreeNode temp;
75             temp = previousNode.getLeft();
76             previousNode.setLeft(null);
77             previousNode = temp;
78             previousNode.setRight(newNode);
79         }
80     }
81     else{

```

```

79         previousNode.setLeft(newNode);
80     }
81     if(newNode.getRight()==null){
82         newNode.setRight(currentNode);
83     }
84     else{
85         if(newNode.getLeft()!=null){ //switch case
86             newNode.getLeft().setRight(currentNode);
87             newNode.setLeft(null);
88         }
89         else if(currentNode!=null){
90             newNode.setLeft(currentNode);
91         }
92     }
93     if(!branchNodeStack.empty() && ((branchNodeStack.peek())[1]) ==
94         previousNode){ //if case
95         if(newNode.getRight()==null){
96             newNode.setRight(branchNodeStack.peek()[0].getLeft());
97         }
98         else{
99             newNode.setLeft(branchNodeStack.peek()[0].getLeft());
100         }
101         branchNodeStack.peek()[0].setLeft(newNode);
102         branchNodeStack.pop();
103     }
104     else if(!branchNodeStack.empty() && newNode.getPcValue() > branchNodeStack.
105         peek()[1].getPcValue()){
106         if(branchNodeStack.peek()[0].getLeft()!=null){
107             branchNodeStack.peek()[1].setRight(branchNodeStack.peek()[0].
108                 getLeft());
109         }
110         branchNodeStack.pop();
111     }
112 }
113
114 public AbstractSyntaxTreeNode findNode(AbstractSyntaxTreeNode root, int pcValue
115 ){
116     AbstractSyntaxTreeNode current = root;
117     AbstractSyntaxTreeNode previous = root;
118     while(current!=null && current.getPcValue() < pcValue){
119         previous = current;
120         if(current.getLeft()!=null && current.getRight()!=null){
121             int leftValue = current.getLeft().getPcValue();
122             int rightValue = current.getRight().getPcValue();
123             if(leftValue <= pcValue && rightValue > pcValue){
124                 current = current.getLeft();
125             }
126             else if(leftValue <= pcValue && rightValue <= pcValue && leftValue

```

```

123         > rightValue){
            current = current.getLeft();
125     }
    else if(rightValue < current.getPcValue()){
127         current = current.getLeft();
    }
    else{
129         current = current.getRight();
    }
131 }
    else if(current.getLeft() == null && current.getRight() != null){
133         current = current.getRight();
    }
135     else{
        current = current.getLeft();
137     }
    }
139     if(current.getPcValue() >= pcValue){
        return previous;
141     }
    else{
143         return current;
    }
145 }

147
private AbstractSyntaxTreeNode popNode(Stack nodeStack){
149     if(!nodeStack.empty()){
        return (AbstractSyntaxTreeNode)nodeStack.pop();
151     }
    else{
153         return null;
    }
155 }

157
public void printTree(Document doc)throws Exception{
159     AbstractSyntaxTreeNode currentNode = this.getRoot();
    currentNode = currentNode.getRight();
161     int indentation = 6; //class and method indentation
    Stack nodeStack = new Stack();
163     while(currentNode!=null){
        if(currentNode.getData().startsWith("}")){ //ends code block
165             indentation -= 3;
        }
        for(int i=0; i<indentation; i++){
167             doc.insertString(doc.getLength(), " ", null);
169         }
    }

```

```

171         if(currentNode.getData().startsWith("if") || currentNode.getData().
            startsWith("else if") || currentNode.getData().startsWith("while")
            || currentNode.getData().startsWith("}while")){ //deleting
173             excessive brackets
            correctBranchNodeData(currentNode);
            optimizeBranchNodeData(currentNode);
175         }
        ReservedWordsPrinter.printLine(currentNode.getData(), doc);
177         if(currentNode.getData().endsWith("{")){ //new code block
            indentation += 3;
179         }
        doc.insertString(doc.getLength(), "\n", null);
181         if(currentNode.getLeft()==null && currentNode.getRight()==null){
            currentNode = popNode(nodeStack);
183         }
        else if(currentNode.getLeft()==null){
            if ((!nodeStack.empty() && nodeStack.peek()==currentNode.getRight())
            ||
185             currentNode.getRight().getPcValue() < currentNode.getPcValue() ||
            (!nodeStack.empty() && currentNode.getRight().getPcValue() > ((
                AbstractSyntaxTreeNode)nodeStack.peek()).getPcValue())){
                currentNode = popNode(nodeStack);
187             }
            else{
189                 currentNode = currentNode.getRight();
            }
191         }
        else if(currentNode.getRight()==null){
            if(!nodeStack.empty() && nodeStack.peek()==currentNode.getLeft() ||
            currentNode.getLeft().getPcValue() < currentNode.getPcValue()){
193                 currentNode = popNode(nodeStack);
            }
            else{
195                 currentNode = currentNode.getLeft();
            }
197         }
        else{
199             int leftValue = currentNode.getLeft().getPcValue();
            int rightValue = currentNode.getRight().getPcValue();
            int currentValue = currentNode.getPcValue();
            if(leftValue < rightValue){
201                 if(leftValue > currentValue){
203                     if(nodeStack.empty() || nodeStack.peek()!=currentNode.
                        getRight()){
205                         nodeStack.push(currentNode.getRight());
                        currentNode = currentNode.getLeft();
207                     }
                }
            }
209         }
211     }

```

```

213         else{
                currentNode = currentNode.getRight();
        }
215     }
    else{
217         if(rightValue > currentValue){
                if(nodeStack.empty() || nodeStack.peek() != currentNode.
                    getLeft()){
219                     nodeStack.push(currentNode.getLeft());
                    currentNode = currentNode.getRight();
221                }
            }
223         else{
                currentNode = currentNode.getLeft();
225            }
        }
227    }
}
229 }

231 protected static void correctBranchNodeData(AbstractSyntaxTreeNode currentNode)
{
    String currentNodeData = currentNode.getData();
233    int leftBrackets = 0;
    int rightBrackets = 0;
235    for(int i = 0; i < currentNodeData.length(); i++){
        if(currentNodeData.charAt(i) == '('){
237            leftBrackets++;
        }
239        else if(currentNodeData.charAt(i) == ')'){
            rightBrackets++;
241        }
    }
243    for(int i=0; i < leftBrackets - rightBrackets; i++){
        if(currentNodeData.startsWith("}while")){
245            currentNodeData = currentNodeData.replace(";", " ");
        }
247        else{
            currentNodeData = currentNodeData.replace("{", "{}");
249        }
    }
251    leftBrackets = 0;
    while((currentNodeData.startsWith("if((") || currentNodeData.startsWith("
        else if((") || currentNodeData.startsWith("while((") || currentNodeData
        .startsWith("}while(("))&& (currentNodeData.contains("))){") ||
        currentNodeData.contains("))")) && leftBrackets == 0){
253        int initialIndex = currentNodeData.indexOf('(') + 2;
        for(int i = initialIndex; i < currentNodeData.length() - 3; i++){

```

```

255         if(currentNodeData.charAt(i) == '('){
256             leftBrackets++;
257         }
258         else if(currentNodeData.charAt(i) == ')') && leftBrackets != 0){
259             leftBrackets--;
260         }
261     }
262     if(leftBrackets == 0){
263         currentNodeData = new StringBuilder(currentNodeData).deleteCharAt(
264             currentNodeData.indexOf('(')).toString();
265         currentNodeData = new StringBuilder(currentNodeData).deleteCharAt(
266             currentNodeData.lastIndexOf('')).toString();
267     }
268     currentNode.setData(currentNodeData);
269 }
270
271 private void optimizeBranchNodeData(AbstractSyntaxTreeNode currentNode){
272     String currentNodeData = currentNode.getData();
273     Stack<Integer []> bracketStack = new Stack<Integer []>();
274     HashSet<Integer> bracketsToDelete = new HashSet<Integer>();
275     ArrayList<Character> data = new ArrayList<Character>();
276     for(int i = 3; i < currentNodeData.length()-2; i++){
277         if(currentNodeData.charAt(i) == '('){
278             if(currentNodeData.charAt(i-2) == '|'){
279                 bracketStack.push(new Integer [] {i, 0, 0, 1});
280             }
281             else{
282                 bracketStack.push(new Integer [] {i, 0, 0, 0});
283             }
284         }
285         else if(currentNodeData.charAt(i) == ')'){
286             if(bracketStack.peek()[1] != 0 && bracketStack.peek()[2] == 0 ||
287                 bracketStack.peek()[1] == 0 && bracketStack.peek()[2] == 0 ||
288                 bracketStack.peek()[3] != 0 && currentNodeData.charAt(i+1) == '
289                 '){
290                 bracketsToDelete.add(bracketStack.peek()[0]);
291                 bracketsToDelete.add(i);
292             }
293             bracketStack.pop();
294         }
295         else if(!bracketStack.empty() && currentNodeData.charAt(i) == '&' &&
296             currentNodeData.charAt(i+1) == '&'){
297             bracketStack.peek()[1] = 1;
298             i++;
299         }
300         else if(!bracketStack.empty() && currentNodeData.charAt(i) == '|' &&
301             currentNodeData.charAt(i+1) == '|'){

```

```

297         bracketStack. peek() [2] = 1;
298         i++;
299     }
300 }
301 for(int i = 0; i < currentNodeData.length(); i++){
302     if(!bracketsToDelete.contains(i)){
303         data.add(currentNodeData.charAt(i));
304     }
305 }
306 currentNodeData = "";
307 for(int i = 0; i < data.size(); i++){
308     currentNodeData = currentNodeData + data.get(i);
309 }
310 currentNode.setData(currentNodeData);
311 }
}

```

```

package jbv;
2
public class AbstractSyntaxTreeNode {
4
    private int pcValue;
6    private String data;
    private AbstractSyntaxTreeNode left;
8    private AbstractSyntaxTreeNode right;

10    AbstractSyntaxTreeNode(int pcValue){
        this.pcValue = pcValue;
12        data = null;
        left = right = null;
14    }

16    AbstractSyntaxTreeNode(int pcValue, String data){
        this.pcValue = pcValue;
18        this.data = data;
        left = right = null;
20    }

22    AbstractSyntaxTreeNode(int pcValue, AbstractSyntaxTreeNode left,
        AbstractSyntaxTreeNode right){
24        this.pcValue = pcValue;
        this.right = right;
        this.left = left;
26    }
}

```

```

28     AbstractSyntaxTreeNode(int pcValue, String data, AbstractSyntaxTreeNode left,
    AbstractSyntaxTreeNode right){
    this.pcValue = pcValue;
30     this.data = data;
    this.right = right;
32     this.left = left;
    }
34
    public void setPcValue(int pcValue){
36         this.pcValue = pcValue;
    }
38
    public int getPcValue(){
40         return this.pcValue;
    }
42
    public void setData(String data){
44         this.data = data;
    }
46
    public String getData(){
48         return this.data;
    }
50
    public void setLeft(AbstractSyntaxTreeNode left){
52         this.left = left;
    }
54
    public AbstractSyntaxTreeNode getLeft(){
56         return this.left;
    }
58
    public void setRight(AbstractSyntaxTreeNode right){
60         this.right = right;
    }
62
    public AbstractSyntaxTreeNode getRight(){
64         return this.right;
    }
66 }

```

```

1  package jbv;
3  import java.util.*;

```

```

5 public class SymbolTable {
7     public static final HashMap<String, String[]> symbolTable = new HashMap<String
        , String[]>();
9     static{
10         symbolTable.put("nop", new String[]{"noAction", null});
11         symbolTable.put("return", new String[]{"noAction", null});
12         symbolTable.put("jsr", new String[]{"noAction", null});
13         symbolTable.put("jsr_w", new String[]{"noAction", null});
14         symbolTable.put("ret", new String[]{"noAction", null});
15         symbolTable.put("return", new String[]{"noAction", null});
16         symbolTable.put("ireturn", new String[]{"return", null});
17         symbolTable.put("lreturn", new String[]{"return", null});
18         symbolTable.put("freturn", new String[]{"return", null});
19         symbolTable.put("dreturn", new String[]{"return", null});
20         symbolTable.put("areturn", new String[]{"return", null});
21         symbolTable.put("wide", new String[]{"noAction", null});
22         symbolTable.put("iconst_m1", new String[]{"push", "-1"});
23         symbolTable.put("iconst_0", new String[]{"push", "0"});
24         symbolTable.put("iconst_1", new String[]{"push", "1"});
25         symbolTable.put("iconst_2", new String[]{"push", "2"});
26         symbolTable.put("iconst_3", new String[]{"push", "3"});
27         symbolTable.put("iconst_4", new String[]{"push", "4"});
28         symbolTable.put("iconst_5", new String[]{"push", "5"});
29         symbolTable.put("iconst", new String[]{"push", "-"});
30         symbolTable.put("lconst_0", new String[]{"push", "0"});
31         symbolTable.put("lconst_1", new String[]{"push", "1"});
32         symbolTable.put("lconst", new String[]{"push", "-"});
33         symbolTable.put("bipush", new String[]{"push", "-"});
34         symbolTable.put("sipush", new String[]{"push", "-"});
35         symbolTable.put("iload", new String[]{"push", null});
36         symbolTable.put("lload", new String[]{"push", null});
37         symbolTable.put("fload", new String[]{"push", null});
38         symbolTable.put("dload", new String[]{"push", null});
39         symbolTable.put("aload", new String[]{"push", null});
40         symbolTable.put("fconst_0", new String[]{"push", "0.0"});
41         symbolTable.put("dconst_0", new String[]{"push", "0.0"});
42         symbolTable.put("fconst_1", new String[]{"push", "1.0"});
43         symbolTable.put("dconst_1", new String[]{"push", "1.0"});
44         symbolTable.put("fconst_2", new String[]{"push", "2.0"});
45         symbolTable.put("iload_0", new String[]{"push", "var0"});
46         symbolTable.put("lload_0", new String[]{"push", "var0"});
47         symbolTable.put("fload_0", new String[]{"push", "var0"});
48         symbolTable.put("dload_0", new String[]{"push", "var0"});
49         symbolTable.put("iload_1", new String[]{"push", "var1"});
50         symbolTable.put("aload_1", new String[]{"push", "var1"});
51         symbolTable.put("lload_1", new String[]{"push", "var1"});

```

```

53     symbolTable.put("fload_1", new String[] {"push", "var1"});
    symbolTable.put("dload_1", new String[] {"push", "var1"});
    symbolTable.put("iload_2", new String[] {"push", "var2"});
55     symbolTable.put("aload_2", new String[] {"push", "var2"});
    symbolTable.put("lload_2", new String[] {"push", "var2"});
57     symbolTable.put("fload_2", new String[] {"push", "var2"});
    symbolTable.put("dload_2", new String[] {"push", "var2"});
59     symbolTable.put("iload_3", new String[] {"push", "var3"});
    symbolTable.put("aload_3", new String[] {"push", "var3"});
61     symbolTable.put("lload_3", new String[] {"push", "var3"});
    symbolTable.put("fload_3", new String[] {"push", "var3"});
63     symbolTable.put("dload_3", new String[] {"push", "var3"});
    symbolTable.put("aload_0", new String[] {"push", "this"});
65     symbolTable.put("dstore_0", new String[] {"store", "double"});
    symbolTable.put("dstore_1", new String[] {"store", "double"});
67     symbolTable.put("dstore_2", new String[] {"store", "double"});
    symbolTable.put("dstore_3", new String[] {"store", "double"});
69     symbolTable.put("fstore_0", new String[] {"store", "float"});
    symbolTable.put("fstore_1", new String[] {"store", "float"});
71     symbolTable.put("fstore_2", new String[] {"store", "float"});
    symbolTable.put("fstore_3", new String[] {"store", "float"});
73     symbolTable.put("istore_0", new String[] {"store", "int"});
    symbolTable.put("istore_1", new String[] {"store", "int"});
75     symbolTable.put("istore_2", new String[] {"store", "int"});
    symbolTable.put("istore_3", new String[] {"store", "int"});
77     symbolTable.put("lstore_0", new String[] {"store", "long"});
    symbolTable.put("lstore_1", new String[] {"store", "long"});
79     symbolTable.put("lstore_2", new String[] {"store", "long"});
    symbolTable.put("lstore_3", new String[] {"store", "long"});
81     symbolTable.put("istore", new String[] {"store", "int"});
    symbolTable.put("lstore", new String[] {"store", "long"});
83     symbolTable.put("fstore", new String[] {"store", "float"});
    symbolTable.put("dstore", new String[] {"store", "double"});
85     symbolTable.put("astore_0", new String[] {"store", null});
    symbolTable.put("astore_1", new String[] {"store", null});
87     symbolTable.put("astore_2", new String[] {"store", null});
    symbolTable.put("astore_3", new String[] {"store", null});
89     symbolTable.put("astore", new String[] {"store", null});
    symbolTable.put("iadd", new String[] {"operation", "+"});
91     symbolTable.put("ladd", new String[] {"operation", "+"});
    symbolTable.put("fadd", new String[] {"operation", "+"});
93     symbolTable.put("dadd", new String[] {"operation", "+"});
    symbolTable.put("isub", new String[] {"operation", "-"});
95     symbolTable.put("lsub", new String[] {"operation", "-"});
    symbolTable.put("fsub", new String[] {"operation", "-"});
97     symbolTable.put("dsub", new String[] {"operation", "-"});
    symbolTable.put("imul", new String[] {"operation", "*"});
99     symbolTable.put("lmul", new String[] {"operation", "*"});

```

```

symbolTable.put("fmul", new String[]{"operation", "*"});
101 symbolTable.put("dmul", new String[]{"operation", "*"});
symbolTable.put("idiv", new String[]{"operation", "/"});
103 symbolTable.put("ldiv", new String[]{"operation", "/"});
symbolTable.put("fdiv", new String[]{"operation", "/"});
105 symbolTable.put("ddiv", new String[]{"operation", "/"});
symbolTable.put("irem", new String[]{"operation", "%"});
107 symbolTable.put("lrem", new String[]{"operation", "%"});
symbolTable.put("frem", new String[]{"operation", "%"});
109 symbolTable.put("drem", new String[]{"operation", "%"});
symbolTable.put("ior", new String[]{"operation", "|"});
111 symbolTable.put("lor", new String[]{"operation", "|"});
symbolTable.put("iand", new String[]{"operation", "&"});
113 symbolTable.put("land", new String[]{"operation", "&"});
symbolTable.put("ixor", new String[]{"operation", "^"});
115 symbolTable.put("lxor", new String[]{"operation", "^"});
symbolTable.put("ishl", new String[]{"operation", "<<"});
117 symbolTable.put("lshl", new String[]{"operation", "<<"});
symbolTable.put("ishr", new String[]{"operation", ">>"});
119 symbolTable.put("lshr", new String[]{"operation", ">>"});
symbolTable.put("iushr", new String[]{"operation", ">>>"});
121 symbolTable.put("lushr", new String[]{"operation", ">>>"});
symbolTable.put("pop", new String[]{"pop", null});
123 symbolTable.put("pop2", new String[]{"pop", null});
symbolTable.put("dup", new String[]{"dup", null});
125 symbolTable.put("dup2", new String[]{"dup", "2"});
symbolTable.put("dup_x1", new String[]{"dup", "x1"});
127 symbolTable.put("dup_x2", new String[]{"dup", "x2"});
symbolTable.put("dup2_x1", new String[]{"dup", "2_x1"});
129 symbolTable.put("dup2_x2", new String[]{"dup", "2_x2"});
symbolTable.put("aconst_null", new String[]{"push", "null"});
131 symbolTable.put("ineg", new String[]{"singleOperation", "~"});
symbolTable.put("iaload", new String[]{"push", "["});
133 symbolTable.put("laload", new String[]{"push", "["});
symbolTable.put("faload", new String[]{"push", "["});
135 symbolTable.put("daload", new String[]{"push", "["});
symbolTable.put("caload", new String[]{"push", "["});
137 symbolTable.put("saload", new String[]{"push", "["});
symbolTable.put("baload", new String[]{"push", "["});
139 symbolTable.put("aaload", new String[]{"push", "["});
symbolTable.put("iastore", new String[]{"store", "["});
141 symbolTable.put("lastore", new String[]{"store", "["});
symbolTable.put("fastore", new String[]{"store", "["});
143 symbolTable.put("dastore", new String[]{"store", "["});
symbolTable.put("aastore", new String[]{"store", "["});
145 symbolTable.put("bastore", new String[]{"store", "["});
symbolTable.put("castore", new String[]{"store", "["});
147 symbolTable.put("sastore", new String[]{"store", "["});

```

```

symbolTable.put("d2f", new String[]{"cast", "float"});
149 symbolTable.put("d2i", new String[]{"cast", "int"});
symbolTable.put("d2l", new String[]{"cast", "long"});
151 symbolTable.put("f2d", new String[]{"cast", "double"});
symbolTable.put("f2i", new String[]{"cast", "int"});
153 symbolTable.put("f2l", new String[]{"cast", "long"});
symbolTable.put("i2b", new String[]{"cast", "byte"});
155 symbolTable.put("i2c", new String[]{"cast", "char"});
symbolTable.put("i2d", new String[]{"cast", "double"});
157 symbolTable.put("i2f", new String[]{"cast", "float"});
symbolTable.put("i2l", new String[]{"cast", "long"});
159 symbolTable.put("i2s", new String[]{"cast", "short"});
symbolTable.put("l2d", new String[]{"cast", "double"});
161 symbolTable.put("l2f", new String[]{"cast", "float"});
symbolTable.put("l2i", new String[]{"cast", "int"});
163 symbolTable.put("if_icmpeq", new String[]{"branch", "=="});
symbolTable.put("if_icmpne", new String[]{"branch", "!="});
165 symbolTable.put("if_icmplt", new String[]{"branch", "<"});
symbolTable.put("if_icmpgt", new String[]{"branch", ">"});
167 symbolTable.put("if_icmple", new String[]{"branch", "<="});
symbolTable.put("if_icmpge", new String[]{"branch", ">="});
169 symbolTable.put("ifeq", new String[]{"branch", " == 0"});
symbolTable.put("ifne", new String[]{"branch", " != 0"});
171 symbolTable.put("iflt", new String[]{"branch", " < 0"});
symbolTable.put("ifgt", new String[]{"branch", " > 0"});
173 symbolTable.put("ifge", new String[]{"branch", " >= 0"});
symbolTable.put("ifle", new String[]{"branch", " <= 0"});
175 symbolTable.put("ifnonnull", new String[]{"branch", " != null"});
symbolTable.put("ifnull", new String[]{"branch", " == null"});
177 symbolTable.put("if_acmpeq", new String[]{"branch", "=="});
symbolTable.put("if_acmpne", new String[]{"branch", "!="});
179 symbolTable.put("fcmpl", new String[]{"branch", null});
symbolTable.put("fcmpg", new String[]{"branch", null});
181 symbolTable.put("dcmpl", new String[]{"branch", null});
symbolTable.put("dcmpg", new String[]{"branch", null});
183 symbolTable.put("lcmpl", new String[]{"branch", null});
symbolTable.put("lcmpg", new String[]{"branch", null});
185 symbolTable.put("iinc", new String[]{"increment", null});
symbolTable.put("newarray", new String[]{"creation", " "});
symbolTable.put("anewarray", new String[]{"creation", "."});
187 symbolTable.put("arraylength", new String[]{"length", null});
symbolTable.put("ldc", new String[]{"ldc", null});
189 symbolTable.put("ldc_w", new String[]{"ldc", null});
symbolTable.put("ldc2_w", new String[]{"ldc", null});
191 symbolTable.put("getfield", new String[]{"getField", null});
symbolTable.put("invokevirtual", new String[]{"invoke", "virtual"});
193 symbolTable.put("invokestatic", new String[]{"invoke", "static"});
symbolTable.put("invokeinterface", new String[]{"invoke", "interface"});
195 symbolTable.put("invokespecial", new String[]{"invoke", "special"});

```

```

symbolTable.put("new", new String[]{ "new", null});
197 symbolTable.put("multianewarray", new String[]{ "multiCreation", null});
symbolTable.put("swap", new String[]{ "swap", null});
199 symbolTable.put("monitorenter", new String[]{ "synchronize", null});
symbolTable.put("monitorexit", new String[]{ "desynchronize", null});
201 symbolTable.put("putfield", new String[]{ "putField", null});
symbolTable.put("putstatic", new String[]{ "putStatic", null});
203 symbolTable.put("getstatic", new String[]{ "getStatic", null});
symbolTable.put("checkcast", new String[]{ "checkcast", null});
205 symbolTable.put("instanceof", new String[]{ "instance", null});
symbolTable.put("tableswitch", new String[]{ "switch", null});
207 symbolTable.put("lookupswitch", new String[]{ "switch", null});
symbolTable.put("athrow", new String[]{ "throw", null});
209 symbolTable.put("goto", new String[]{ "branch", "goto"});
symbolTable.put("goto_w", new String[]{ "branch", "goto"});
211 }
213 }

```

```

1 package jbv;

3 import javax.swing.text.*;
import java.awt.Color;
5 import java.util.*;

7 public class ReservedWordsPrinter {

9     private static final StyleContext context = new StyleContext();
private static final Style reservedWordStyle = context.addStyle("
    reservedWordStyle", null);
11 private static final Style stringStyle = context.addStyle("stringStyle", null);
private static final float[] darkGreen = Color.RGBtoHSB(255, 140, 0, null);
13

static{
15     reservedWordStyle.addAttribute(StyleConstants.Foreground, Color.BLUE);
stringStyle.addAttribute(StyleConstants.Foreground, Color.getHSBColor(
    darkGreen[0], darkGreen[1], darkGreen[2]));
17 }

19 public static Set<String> keywords = new HashSet<String> (Arrays.asList(
    "assert",
21     "abstract", "boolean", "break", "byte",
    "case", "catch", "char", "class",
23     "const", "continue", "default", "do",
    "double", "else", "extends", "false", "final",
25     "finally", "float", "for", "goto",

```

```

27     "if", "implements", "import",
28     "instanceof", "int", "interface",
29     "long", "native", "new", "package",
30     "private", "protected", "public",
31     "return", "short", "static", "super",
32     "switch", "synchronized", "this",
33     "throw", "throws", "transient", "true",
34     "try", "void", "volatile", "while"));
35
36 public static void printLine(String line, Document doc)throws Exception{
37
38     //deal with comments
39     //deal with strings
40     StringBuilder word = new StringBuilder(0);
41     char[] lineArray = line.toCharArray();
42     char previousChar=' ', nextChar=' ';
43     Boolean isString = false; //for string printing in decompiling
44     int slashOccurencies = 0;
45     for(int i=0; i<lineArray.length; i++){
46         if((lineArray[i] == '"') && ((lineArray[i-1]!='\\') || (lineArray[i+1]!='\\')) && (slashOccurencies % 2 == 0)){
47             if(isString == false){
48                 isString = true;
49             }
50             else{
51                 isString = false;
52             }
53             previousChar = '"'; // for typical reason but is is unnecessary
54             doc.insertString(doc.getLength(),String.valueOf(lineArray[i]),
55                 stringStyle);
56             slashOccurencies = 0;
57         }
58         else if(lineArray[i] == '\\'){
59             slashOccurencies++;
60             doc.insertString(doc.getLength(),String.valueOf(lineArray[i]),
61                 stringStyle);
62             previousChar = '\\';
63         }
64         else if(isString == true){
65             doc.insertString(doc.getLength(),String.valueOf(lineArray[i]),
66                 stringStyle);
67             slashOccurencies = 0;
68         }
69         else if (Character.isLetter(lineArray[i])){
70             word.append(lineArray[i]);
71             slashOccurencies = 0;
72         }
73         else{

```

```

71         nextChar = lineArray[i];
72         if(word.length() != 0){
73             if(isKeyword(word.toString(), previousChar, nextChar)){
74                 doc.insertString(doc.getLength(), word.toString(),
75                     reservedWordStyle);
76             }
77             else{
78                 doc.insertString(doc.getLength(), word.toString(), null);
79             }
80             previousChar = nextChar;
81             doc.insertString(doc.getLength(), String.valueOf(nextChar), null);
82             word.setLength(0);
83             slashOccurencies = 0;
84         }
85     }
86 }
87
88 private static Boolean isKeyword(String word, char previous, char next){
89     Boolean isKeyword = false;
90     if((previous != '}' ) && (previous != '(') && (previous != ' ') && (previous != '\t'
91         ' ) && (previous != '.' ) && (previous != '=' ) && (previous != '/')){ //a
92         keyword is precided by space \t } ( . = /
93         return false;
94     }
95     if((next != '(') && (next != '{') && (next != '[') && (next != ':') && (
96         next != ';' ) && (next != ' ') && (next != '\t') && (next != '.' ) && (next != '/')
97     ){ // a keyword is followed by { [ space \t ; : /.
98         return false;
99     }
100     if(keywords.contains(word)){
101         isKeyword = true;
102     }
103     return isKeyword;
104 }

```

```

1 package jbv;
2
3 import java.awt.*;
4 import javax.swing.*;
5
6 public class Main {

```

```

7      private JTextPane sourceCode;
9
10     private JTextPane byteCode;
11
12     private actionPerformed action;
13
14     private JFrame frame;
15
16
17     public JTextPane getTextArea(){
18         return this.sourceCode;
19     }
20
21     public JTextPane getBytecodeTextArea(){
22         return this.byteCode;
23     }
24
25     public actionPerformed getAction(){
26         return action;
27     }
28
29     public JFrame getFrame(){
30         return frame;
31     }
32
33     public static void main(String[] args) {
34
35         Main m = new Main();
36         m.frame = new JFrame();
37         m.frame.setResizable(false);
38         m.frame.setTitle("JBVD - Java Bytecode Viewer & Decompiler - Developed By
39             Nicole Tryfona");
40         Container container = m.frame.getContentPane();
41         GridBagLayout gbl = new GridBagLayout();
42         container.setLayout(gbl);
43         JPanel classPanel = new JPanel(new BorderLayout());
44         classPanel.setBorder(BorderFactory.createTitledBorder("Class Source Code"))
45             ;
46         classPanel.setPreferredSize(new Dimension(500,500));
47         JPanel bytecodePanel = new JPanel(new BorderLayout());
48         bytecodePanel.setBorder(BorderFactory.createTitledBorder("Class Bytecode"))
49             ;
50         bytecodePanel.setPreferredSize(new Dimension(500,500));
51         //Scrolling is controlled by the implementation of the Scrollable interface
52         .
53         //By overriding the getScrollableTracksViewportWidth() method we can
54         achieve a

```

```

51 //non-wrapping text pane:
m.sourceCode = new JTextPane(){
    @Override
53     public boolean getScrollableTracksViewportWidth()
    {
55         return getUI().getPreferredSize(this).width
            <= getParent().getSize().width;
57     }
};
59 m.sourceCode.setEditable(false);
JScrollPane pane1 = new JScrollPane(m.sourceCode);
61 pane1.setMinimumSize(new Dimension(487,467));
classPanel.add(pane1);
63 m.byteCode = new JTextPane(){
    @Override
65     public boolean getScrollableTracksViewportWidth()
    {
67         return getUI().getPreferredSize(this).width
            <= getParent().getSize().width;
69     }
};
71 m.byteCode.setEditable(false);
JScrollPane pane2 = new JScrollPane(m.byteCode);
73 pane2.setMinimumSize(new Dimension(487, 467));
bytecodePanel.add(pane2);
75 container.add(classPanel);
container.add(bytecodePanel);
77 JButton button = new JButton("File Selection");
m.action = new actionPerformed(m);
79 button.addActionListener(m.action);
GridBagConstraints c = new GridBagConstraints();
81 c.gridx = 0;
c.gridwidth = 2;
83 c.weighty = 4;
c.anchor = GridBagConstraints.CENTER;
85 c.insets = new Insets(10,0,10,0);
container.add(button, c);
87 m.frame.pack();
m.frame.setVisible(true);
89 m.frame.setLocationRelativeTo(null);
m.frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
91 }
93 }

```

```

1 package jbv;

```

```

3  import java.io.*;

5  public class SourceCodeFileFilter extends javax.swing.filechooser.FileFilter{

7      private final String[] okFileExtensions = new String[] { "class", "java" };

9      SourceCodeFileFilter(){

11         }

13     public boolean accept(File file){

15         for(String extension : okFileExtensions){
16             if(file.getName().toLowerCase().endsWith(extension)){
17                 return true;
18             }
19             else if(file.isDirectory()){
20                 return true;
21             }
22         }
23         return false;
24     }

25     public String getDescription(){
26         return "Java Source Code Files";
27     }
28 }

```

```

package jbv;

2  import java.awt.event.*;
4  import javax.swing.*;
5  import java.awt.*;
6  import java.io.*;
7  import javax.swing.text.*;

8

10 public class actionToPerform implements ActionListener{

12     private static final StyleContext context = new StyleContext();
13     private static final Style commentStyle = context.addStyle("reservedWordStyle",
14         null);
15     private static final Style stringStyle = context.addStyle("stringStyle", null);
16     private static Boolean multiLineComment = false;

```

```

16     private static final float [] darkGreen = Color.RGBtoHSB(255, 140, 0, null);

18     static{
19         commentStyle.addAttribute(StyleConstants.Foreground, Color.GRAY);
20         stringStyle.addAttribute(StyleConstants.Foreground, Color.getHSBColor(
21             darkGreen[0], darkGreen[1], darkGreen[2]));
22     }

23     private Main m;

24     private String filePath;

25     private String parentFilePath;

26     private String fileName;

27     private String packageName = null;

28     public actionToPerform(Main m){
29         this.m = m;
30     }

31     public String getFilePath(){
32         return filePath;
33     }

34     public String getParentFilePath(){
35         return parentFilePath;
36     }

37     public String getFileName(){
38         return fileName;
39     }

40     public String getPackageName(){
41         return packageName;
42     }

43     public void actionPerformed(ActionEvent e){
44         JFileChooser fc = new JFileChooser();
45         SourceCodeFileFilter filter = new SourceCodeFileFilter();
46         fc.setFileFilter(filter);
47         fc.setAcceptAllFileFilterUsed(false);
48         JFrame frame2 = new JFrame();
49         Container container2 = frame2.getContentPane();
50         GridBagLayout gbl2 = new GridBagLayout();
51         container2.setLayout(gbl2);
52         JPanel fcPanel = new JPanel(new GridBagLayout());

```

```

64      int returnVal = fc.showOpenDialog(fcPanel);
Document doc1 = m.getTextArea().getDocument();
Document doc2 = m.getBytecodeTextArea().getDocument();
66      if(returnVal == 0){
          m.getTextArea().setText("");
68          m.getBytecodeTextArea().setText("");
          filePath = fc.getSelectedFile().getAbsolutePath().replace("\\", "/");
70          parentFilePath = fc.getSelectedFile().getParent().replace("\\", "/");
          int index = fc.getSelectedFile().getName().lastIndexOf('.');
72          fileName = fc.getSelectedFile().getName().substring(0, index);
          File chosenFile = fc.getSelectedFile();
74          m.getTextArea().setText("");
          if(chosenFile!=null){
76              try{
                  if(fc.getSelectedFile().getAbsolutePath().toLowerCase().endsWith("
                      java")){
78                      FileReader fr = new FileReader(chosenFile);
                      BufferedReader br = new BufferedReader(fr);
80                      packageName = null;
                      int indentation = 0;
82                      String line = br.readLine();
                      if(line.lastIndexOf("{") == line.length() - 1){
84                          indentation += 3;
                      }
86                      while(line != null){
                          String line2 = line; //attempting to get package name
88                          if(line2.indexOf("package") == 0){
                              int ch = 7; // in order to move to the next character
                                  after the word package
90                              while(line2.charAt(ch) == 32){
                                  ch++;
92                              }
                              int initIndex = ch;
94                              while (line2.charAt(ch) != ' ' && line2.charAt(ch) != ';' )
                                  {
                                      ch++;
96                                  }
                              int endIndex = ch;
98                              packageName = line2.substring(initIndex, endIndex);
                          }
100                         if((line.indexOf(' ') == -1) && (line.indexOf('\'') == -1)){
                            print(line, doc1);
102                         }
104                         else if((line.indexOf('"') != -1) && ((line.charAt(line.
                            indexOf('"') - 1) != '\') || (line.charAt(line.indexOf('
                            ') + 1) != '\'))){
                            String leftSubstring = line.substring(0, line.indexOf('
                                '));

```

```

106         if((multiLineComment == true) || (leftSubstring.indexOf(
            "/" != -1) || ((leftSubstring.indexOf("/") != -1) && (
            leftSubstring.indexOf("/") == -1))) {
108             print(line, doc1);
110         }
112         else {
            print(leftSubstring, doc1);
            line = line.substring(line.indexOf('')+1);
            if((line.indexOf('')-1 < 0) || (line.charAt(line.
                indexOf('')-1) != '\\') || ((line.charAt(line.
                indexOf('')-1) == '\\') && (line.charAt(line.
                indexOf('\\')+1) == '\\') /* case "\\\" */)) {
114                 doc1.insertString(doc1.getLength(), ''+line.
                    substring(0, line.indexOf('')+1),
                    stringStyle);
            }
            else {
116                 doc1.insertString(doc1.getLength(), "\"",
                    stringStyle);
118                 int i = line.indexOf('');
120                 while(i > 0 && line.charAt(i-1) == '\\') {
                    doc1.insertString(doc1.getLength(), line.
                        substring(0, i+1), stringStyle);
                    line = line.substring(i+1);
                    i = line.indexOf('');
                }
122                 doc1.insertString(doc1.getLength(), line.
                    substring(0, i+1), stringStyle);
124             }
126             line = line.substring(line.indexOf('')+1);
            continue; // in order to check for new strings in
                       the right substring
128         }
130     }
132     else if(line.indexOf('\\') != -1) {
        String leftSubstring = line.substring(0, line.indexOf(
            '\\'));
134         if((multiLineComment == true) || (leftSubstring.indexOf(
            "/" != -1) || ((leftSubstring.indexOf("/") != -1) && (
            leftSubstring.indexOf("/") == -1))) {
            print(line, doc1);
136         }
138         else {
            print(leftSubstring, doc1);
            line = line.substring(line.indexOf('\\')+1);
            doc1.insertString(doc1.getLength(), '\\' + line.

```

```

140         substring(0, line.indexOf('\n')+1), stringStyle)
141     ;
142     line = line.substring(line.indexOf('\n')+1);
143     continue; // in order to check for new strings in
144     the right substring
145 }
146 }
147 doc1.insertString(doc1.getLength(), "\n", null);
148 line = br.readLine();
149 if(line != null){
150     line = line.trim();
151     if(line.lastIndexOf("}") == line.length() - 1 || line.
152     charAt(0) == '{'){
153         indentation -= 3;
154     }
155     for(int i = 0; i < indentation; i++){
156         doc1.insertString(doc1.getLength(), " ", null);
157     }
158     if(line.lastIndexOf("{") == line.length() - 1){
159         indentation += 3;
160     }
161 }
162 }
163 BytecodeConversion bc = new BytecodeConversion(m);
164 if(fc.getSelectedFile().getAbsolutePath().toLowerCase().endsWith("
165 java")){
166     bc.convertToBytecode(doc2);
167 }
168 else{
169     JavaConversion jc = new JavaConversion(m);
170     bc.convertToBytecode(doc2);
171     jc.convertToJava();
172 }
173 }
174 catch (FileNotFoundException fnfe){
175 }
176 catch (IOException ioe){
177 }
178 }
179 }
180 private void print(String line, Document doc1) throws Exception{
181     if((line.indexOf("/")!=-1) && (line.indexOf("/")!=0) && (line.indexOf("*/

```

```

182         ")!=-1)){
        print(line.substring(0, line.indexOf("/")), doc1);
        print(line.substring(line.indexOf("/"),line.indexOf("*/")+2), doc1);
184         print(line.substring(line.indexOf("*/")+2), doc1);
    }
186     else if(line.indexOf("//")!= -1){ //single line comments
        ReservedWordsPrinter.println(line.substring(0, line.indexOf("//")),
            doc1);
188         int index = line.indexOf("//")-1;
        doc1.insertString(doc1.getLength(), line.substring(index < 0 ? 0 :
            index), commentStyle);
190     }
    else if(line.indexOf("/")!=-1 && line.indexOf("*/")==-1){ //multiline
        comments
192         multiLineComment = true;
        ReservedWordsPrinter.println(line.substring(0, line.indexOf("/")),
            doc1);
194         int index = line.indexOf("/")+1;
        doc1.insertString(doc1.getLength(), line.substring(index < 0 ? 0 :
            index), commentStyle);
196     }
    else if (line.indexOf("*/")!= -1){
198         multiLineComment = false;
        doc1.insertString(doc1.getLength(), line.substring(0,line.indexOf("*/")+2), commentStyle);
200         ReservedWordsPrinter.println(line.substring(line.indexOf("*/")+2),
            doc1);
    }
202     else{
        if(multiLineComment == false){
204             ReservedWordsPrinter.println(line, doc1);
        }
        else{
206             doc1.insertString(doc1.getLength(), line, commentStyle);
208         }
    }
210 }
212 }
}

```

```

1 package jbv;
3 import javassist.*;
import java.util.*;
5 import javassist.bytecode.*;

```

```

import javax.tools.*;
7 import javax.tools.DiagnosticCollector;
import javax.tools.Diagnostic;
9 import java.io.*;
import javax.swing.*;
11 import javax.swing.text.*;
import java.awt.Color;
13
public class BytecodeConversion {
15     private Main m;

17     private static final StyleContext context = new StyleContext();
private static final Style headingStyle = context.addStyle("headingStyle", null
    );
19     private static final Style counterStyle = context.addStyle("counterStyle", null)
    ;
private static final Style nestedClassNameStyle = context.addStyle("
    nestedClassNameStyle", null);
21     private static final Style compilationErrorsHeadingStyle = context.addStyle("
    compilationErrorHeadingStyle", null);
private static final float[] darkGreen = Color.RGBtoHSB(0, 100, 0, null);
23
static{
25     headingStyle.addAttribute(StyleConstants.Foreground, Color.RED);
    counterStyle.addAttribute(StyleConstants.Foreground, Color.getHSBColor(
        darkGreen[0], darkGreen[1], darkGreen[2]));
27     nestedClassNameStyle.addAttribute(StyleConstants.Underline, true);
    compilationErrorsHeadingStyle.addAttribute(StyleConstants.Bold, true);
29     compilationErrorsHeadingStyle.addAttribute(StyleConstants.Underline, true);
    compilationErrorsHeadingStyle.addAttribute(StyleConstants.FontSize, 13);
31 }

33     public BytecodeConversion(Main m){
        this.m = m;
35     }

37     public void getNestedClassBytecodes(String name, Document doc) throws Exception{
        doc.insertString(doc.getLength(), "Nested Class Name: "+ name+"\n\n",
            nestedClassNameStyle);
39         ClassPool cp = ClassPool.getDefault();
        cp.insertClassPath(m.getAction().getParentFilePath());
41         CtClass cc = cp.get(name);
        ClassFile cf = cc.getClassFile();
43         CtMethod[] dmethods = cc.getDeclaredMethods();
        int size2 = dmethods.length;
45         doc.insertString(doc.getLength(), "General Information: \n\n", headingStyle
            );
        doc.insertString(doc.getLength(), "Major Version:\t"+ cf.getMajorVersion()+

```

```

        "\nMinor Version:\t"+cf.getMinorVersion() + "\nSuperclass:\t" + cf.
        getSuperclass()+ "\nModifiers:\t"+ Integer.toHexString(cf.
        getAccessFlags()+ "\n\n", null);
47 List list = cf.getFields();
    int n = list.size();
49 doc.insertString(doc.getLength(), "Fields:\n\n", headingStyle);
    for(int i=0; i<n; i++){
51         FieldInfo finfo = (FieldInfo)list.get(i);
        int acc = finfo.getAccessFlags();
53         doc.insertString(doc.getLength(), "Access Flag:\t"+Modifier.toString(
            AccessFlag.toModifier(acc))+ "\tName:\t"+finfo.getName()+ "\t\
            tDescriptor:\t"+finfo.getDescriptor()+ "\n", null);
    }
55 String[] infs = cf.getInterfaces();
    if(infs!=null && infs.length > 0){
57         doc.insertString(doc.getLength(), "Interfaces:\t"+infs[0], null);
        for (int i = 1; i < infs.length; ++i){
59             doc.insertString(doc.getLength(), ", "+infs[i], null);
        }
61     }
    list = cf.getMethods();
63 n = list.size();
    doc.insertString(doc.getLength(), "\nMethods Information:\n\n",
        headingStyle);
65 for (int i = 0; i < n; ++i) {
        MethodInfo minfo = (MethodInfo)list.get(i);
67         int acc = minfo.getAccessFlags();
        doc.insertString(doc.getLength(), Modifier.toString(AccessFlag.
            toModifier(acc))+ " " + minfo.getName() + "\t"+ minfo.getDescriptor
            ()+"\t", null);
69         CodeAttribute ca = minfo.getCodeAttribute();
        doc.insertString(doc.getLength(), "max stack: " + ca.getMaxStack()+ ",
            max locals: " + ca.getMaxLocals()+ ", " + " catch blocks: "+ca.
            getExceptionTable().size()+ "\n", null);
71     }
    doc.insertString(doc.getLength(), "\n", null);
73 for(int i=0; i<size2; i++){
        MethodInfo minfo = cf.getMethod(dmethods[i].getName());
75         doc.insertString(doc.getLength(), "Bytecode of " + dmethods[i].getName
            () + " method:\n\n", headingStyle);
        CodeAttribute ca = minfo.getCodeAttribute();
77         ConstPool pool = minfo.getConstPool();
        CodeIterator ci = ca.iterator();
79         while(ci.hasNext()){
            int index = ci.next();
81             String instruction = InstructionPrinter.instructionString(ci,index,
                pool);
            instruction = JavaConversion.escapeCharacters(instruction);

```

```

83         doc.insertString(doc.getLength(), InstructionPrinter.
            instructionString(ci,index,pool), null);
84         doc.insertString(doc.getLength(), "\n", null);
85     }
86     doc.insertString(doc.getLength(), "\n", null);
87     doc.insertString(doc.getLength(), "\n", null);
88 }
89 CtConstructor[] constructors = cc.getDeclaredConstructors();
90 for(int i=0;i<constructors.length;i++){
91     MethodInfo minfo = constructors[i].getMethodInfo();
92     doc.insertString(doc.getLength(), "Bytecode of " + constructors[i].
        getLongName()+ " (constructor):\n\n", headingStyle);
93     CodeAttribute ca = minfo.getCodeAttribute();
94     ConstPool pool = minfo.getConstPool();
95     CodeIterator ci = ca.iterator();
96     while(ci.hasNext()){
97         int index = ci.next();
98         String instruction = InstructionPrinter.instructionString(ci,index,
        pool);
99         instruction = JavaConversion.escapeCharacters(instruction);
100        doc.insertString(doc.getLength(), instruction, null);
101        doc.insertString(doc.getLength(), "\n", null);
102    }
103    doc.insertString(doc.getLength(), "\n", null);
104 }
105 CtClass[] innerClasses = cc.getNestedClasses();
106 if(innerClasses.length >0){
107     CtClass innerClass;
108     doc.insertString(doc.getLength(), "Nested Classes of " + cc.getName()+
        "class:\n\n", headingStyle);
109     for(int i=0; i<innerClasses.length; i++){
110         innerClass = cp.get(innerClasses[i].getName());
111         getNestedClassBytecodes(innerClasses[i].getName(),doc);
112     }
113 }
114
115
116 }
117
118
119 public void convertToBytecode(Document doc) throws Exception{
120     doc.insertString(doc.getLength(), "", null);
121     doc.remove(0, doc.getLength());
122     boolean compilationErrors = false;
123     if(m.getAction().getFilePath().toLowerCase().endsWith(".java")){
124         compilationErrors = compileSourceCode(doc);
125     }
126     if(compilationErrors == false){

```

```

127     ClassPool cp = new ClassPool();
128     cp.insertClassPath(m.getAction().getParentFilePath());
129     CtClass cc = cp.get(m.getAction().getFileName());
130     ClassFile cf = cc.getClassFile();
131     CtMethod[] dmethods = cc.getDeclaredMethods();
132     doc.insertString(doc.getLength(), "General Information:\n\n",
        headingStyle);
133     doc.insertString(doc.getLength(), "Major Version:\t" + cf.
        getMajorVersion() + "\nMinor Version:\t" + cf.getMinorVersion() + "
        \nSuperclass:\t" + cf.getSuperclass() + "\nModifiers:\t" + Integer.
        toHexString(cf.getAccessFlags()) + "\n\n", null);
134     List list = cf.getFields();
135     int n = list.size();
136     doc.insertString(doc.getLength(), "Fields:\n\n", headingStyle);
137     for(int i = 0; i < n; i++){
138         FieldInfo finfo = (FieldInfo) list.get(i);
139         int acc = finfo.getAccessFlags();
140         doc.insertString(doc.getLength(), "Access Flag:\t" + Modifier.
            toString(AccessFlag.toModifier(acc)) + "\tName:\t" + finfo.
            getName() + "\tDescriptor:\t" + finfo.getDescriptor() + "\n",
            null);
141     }
142     String[] infs = cf.getInterfaces();
143     if(infs != null && infs.length > 0){
144         doc.insertString(doc.getLength(), "Interfaces:\t" + infs[0], null);
145         for(int i = 1; i < infs.length; ++i){
146             doc.insertString(doc.getLength(), ", " + infs[i], null);
147         }
148     }
149     list = cf.getMethods();
150     n = list.size();
151     doc.insertString(doc.getLength(), "\nMethods Information:\n\n",
        headingStyle);
152     for(int i = 0; i < n; ++i){
153         MethodInfo minfo = (MethodInfo) list.get(i);
154         int acc = minfo.getAccessFlags();
155         String access = Modifier.toString(AccessFlag.toModifier(acc));
156         doc.insertString(doc.getLength(), Modifier.toString(AccessFlag.
            toModifier(acc)) + " " + minfo.getName() + "\t" + minfo.
            getDescriptor() + "\t", null);
157         CodeAttribute ca = minfo.getCodeAttribute();
158         if(access.indexOf("abstract") == -1){
159             doc.insertString(doc.getLength(), "max stack: " + ca.
                getMaxStack() + ", max locals: " + ca.getMaxLocals() + ",
                " + " catch blocks: " + ca.getExceptionTable().size() +
                "\n", null);
160         }
161     }
    else{

```

```

163         doc.insertString(doc.getLength(), "\n", null);
164     }
165     doc.insertString(doc.getLength(), "\n", null);
166     for(int i = 0; i < n; i++){
167         MethodInfo minfo = (MethodInfo) list.get(i);
168         int acc = minfo.getAccessFlags();
169         String access = Modifier.toString(AccessFlag.toModifier(acc));
170         if(access.indexOf("abstract") == -1){
171             if(!minfo.getName().equalsIgnoreCase("<init>")){
172                 doc.insertString(doc.getLength(), "Bytecode of " + minfo.
173                     getName() + " method:\n\n", headingStyle);
174                 CodeAttribute ca = minfo.getCodeAttribute();
175                 ConstPool pool = minfo.getConstPool();
176                 CodeIterator ci = ca.iterator();
177                 int currentPC = 0;
178                 while(ci.hasNext()){
179                     doc.insertString(doc.getLength(), currentPC + "    ",
180                         counterStyle);
181                     int index = ci.next();
182                     String instruction = InstructionPrinter.
183                         instructionString(ci, index, pool);
184                     if(instruction.indexOf('\n') != -1){
185                         instruction = JavaConversion.escapeCharacters(
186                             instruction);
187                     }
188                     if(instruction.startsWith("wide")){
189                         doc.insertString(doc.getLength(), "wide \n", null);
190                         currentPC = currentPC + JavaConversion.calculatePC(
191                             "wide", "wide", currentPC);
192                         doc.insertString(doc.getLength(), currentPC + "
193                             ", counterStyle);
194                         instruction = instruction.substring(5);
195                     }
196                     String bytecode = instruction;
197                     if(bytecode.indexOf(" ") != -1){
198                         bytecode = instruction.substring(0, instruction.
199                             indexOf(" "));
200                     }
201                     currentPC = currentPC + JavaConversion.calculatePC(
202                         bytecode, instruction, currentPC);
203                     doc.insertString(doc.getLength(), instruction, null);
204                     doc.insertString(doc.getLength(), "\n", null);
205                 }
206                 doc.insertString(doc.getLength(), "\n", null);
207                 doc.insertString(doc.getLength(), "\n", null);
208             }
209         }
210     }
211 }

```

```

203     }
204     CtConstructor[] constructors = cc.getDeclaredConstructors();
205     for(int i = 0; i < constructors.length; i++){
206         MethodInfo minfo = constructors[i].getMethodInfo();
207         doc.insertString(doc.getLength(), "Bytecode of " + constructors[i].
            getLongName() + " (constructor):\n\n", headingStyle);
208         CodeAttribute ca = minfo.getCodeAttribute();
209         ConstPool pool = minfo.getConstPool();
210         CodeIterator ci = ca.iterator();
211         int currentPC = 0;
212         while(ci.hasNext()){
213             doc.insertString(doc.getLength(), currentPC + "    ",
                counterStyle);
214             int index = ci.next();
215             String instruction = InstructionPrinter.instructionString(ci,
                index, pool);
216             instruction = JavaConversion.escapeCharacters(instruction);
217             String bytecode = instruction;
218             if(bytecode.indexOf(" ") != -1){
219                 bytecode = instruction.substring(0, instruction.indexOf(" "
                    ));
220             }
221             currentPC = currentPC + JavaConversion.calculatePC(bytecode,
                instruction, currentPC);
222             doc.insertString(doc.getLength(), instruction, null);
223             doc.insertString(doc.getLength(), "\n", null);
224         }
225         doc.insertString(doc.getLength(), "\n", null);
226     }
227     CtClass[] innerClasses = cc.getNestedClasses();
228     if(innerClasses.length > 0){
229         CtClass innerClass;
230         doc.insertString(doc.getLength(), "Nested Classes of " + cc.getName
            () + " class:\n\n", nestedClassNameStyle);
231         for(int i = 0; i < innerClasses.length; i++){
232             innerClass = cp.get(innerClasses[i].getName());
233             getNestedClassBytecodes(innerClasses[i].getName(), doc);
234         }
235     }
236 }
237
238
239 private boolean compileSourceCode(Document doc) throws Exception{
240     JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
241     String fp = m.getAction().getFilePath();
242     List<File> filesList = new ArrayList<File>();
243     filesList.add(new File(fp));

```

```

245 List<File> clp = new ArrayList<File>();
File path = new File(m.getAction().getParentFilePath()); // path is C:\
String flag = path.toString(); //flag is C:\
247 String pack = m.getAction().getPackageName(); //package is null
boolean compilationErrors = false;
249 while(flag != null){
    path = new File(flag);
251 clp.add(path);
    if((path.getName().equals(pack) || pack == null)){
253         if(path.getParentFile() != null){
            clp.add(path.getParentFile());
255         }
        break;
257     }
    flag = path.getParent();
259 }
int l = 0;
261 do{
    l = 0;
263 DiagnosticCollector<JavaFileObject> diagnostics = new
        DiagnosticCollector<JavaFileObject>();
    StandardJavaFileManager fileManager = compiler.getStandardFileManager(
        diagnostics, null, null);
265 Iterable<? extends JavaFileObject> compilationUnits = fileManager.
        getJavaFileObjectsFromFiles(filesList);
    fileManager.setLocation(StandardLocation.CLASS_PATH, clp);
267 compiler.getTask(null, fileManager, diagnostics, null, null,
        compilationUnits).call();
    ArrayList<String> errors = new ArrayList(0);
269 ArrayList<String> warnings = new ArrayList(0);
    for(Diagnostic diagnostic : diagnostics.getDiagnostics()){
271         int appear = 1;
        if(diagnostic.getKind() != Diagnostic.Kind.ERROR){
273             warnings.add(diagnostic.toString());
        }
        else{
275             compilationErrors = true;
            String error = diagnostic.toString();
277             if(error.indexOf("cannot find symbol") != -1){
                if(error.indexOf("symbol : class ") != -1){
279                     l++;
                    int startIndex = error.indexOf("symbol : class ") +
                        16;
                    int i = 0;
                    int endIndex = 0;
283                     while (error.charAt(startIndex + i) != '\n') {
                        endIndex = startIndex + i;
285                     i++;

```

```

287         }
288         String missingClass = error.substring(startIndex,
289             endIndex + 1);
290         String missingFilePath = filesList.get(0).getParent() +
291             "\\\" + missingClass + \".java\";
292         filesList.add(new File(missingFilePath));
293         fileManager.close();
294         appear = 0;
295     }
296 }
297
298     if(appear == 1){
299         errors.add(error);
300     }
301 }
302 if(errors.size() > 0){
303     JOptionPane jop = new JOptionPane();
304     jop.showMessageDialog(m.getFrame(), "Compilation errors detected!")
305     ;
306     doc.insertString(doc.getLength(), "The following compilation errors
307         occurred:" + "\n\n", compilationErrorsHeadingStyle);
308     for(int i = 0; i < errors.size(); i++){
309         doc.insertString(doc.getLength(), errors.get(i) + "\n\n",
310             headingStyle);
311     }
312 }
313 }while(l != 0);
314 return compilationErrors;
315 }
316 }

```

```

package jbv;

2
import javassist.*;
4 import javassist.bytecode.*;
import java.util.*;
6 import java.lang.reflect.Method;
import javax.swing.text.*;
8 import java.awt.Color;

10 public class JavaConversion {

12     private Main m;

14     private static final StyleContext context = new StyleContext();

```

```

16     private static final Style reservedWordStyle = context.addStyle("
        reservedWordStyle", null);
16     private static final HashMap<String, String> dataTypes = new HashMap<String,
        String>();

18     static{
        reservedWordStyle.addAttribute(StyleConstants.Foreground, Color.BLUE);
20         dataTypes.put("I", "int");
        dataTypes.put("J", "long");
22         dataTypes.put("L", "long");
        dataTypes.put("D", "double");
24         dataTypes.put("F", "float");
        dataTypes.put("S", "short");
26         dataTypes.put("B", "byte");
        dataTypes.put("Z", "boolean");
28         dataTypes.put("C", "char");
    }

30
32     private HashMap<Integer, Object[]> varTypes = new HashMap<Integer, Object[]>();
    private LinkedHashMap<Integer, String> instructions = new LinkedHashMap<Integer
        , String>();
    private Iterator iterator;
34     private int currentPC;
    private int currentVar;
36     private int currentException;
    private int exceptionIndex;
38     private Stack operandStack = new Stack();
    private Stack<Integer> lowerPCValues = new Stack<Integer>();
40     private ArrayList<Integer> rightBracketList = new ArrayList<Integer>();
    private Stack<AbstractSyntaxTreeNode[]> branchNodeStack = new Stack<
        AbstractSyntaxTreeNode[]>();
42     private Stack<AbstractSyntaxTreeNode[]> gotoNodeStack = new Stack<
        AbstractSyntaxTreeNode[]>();
    private HashMap<Integer, AbstractSyntaxTreeNode> breakNodes = new HashMap<
        Integer, AbstractSyntaxTreeNode>();
44     private LinkedList<AbstractSyntaxTreeNode[]> caseNodeList = new LinkedList<
        AbstractSyntaxTreeNode[]>();
    private String currentInstruction = null;
46     private String previousInstruction = null;
    private String bytecode = null;
48     private String action = null;
    private String symbol = null;
50     private AbstractSyntaxTree tree;
    private AbstractSyntaxTreeNode previousNode;

52
54     public JavaConversion(Main m){
        this.m = m;
    }

```

```

56 public void convertToJava() throws Exception{
57     Document doc = m.getTextArea().getDocument();
58     ClassPool cp = new ClassPool();
59     cp.insertClassPath(m.getAction().getParentFilePath());
60     CtClass cc = cp.get(m.getAction().getFileName());
61     ClassFile cf = cc.getClassFile();
62     String className = cf.getName();
63     String packageName = " ";
64     varTypes = new HashMap<Integer, Object[]>();
65     while(className.indexOf(".")!=-1){
66         packageName = packageName.concat(className.substring(0,className.
67             indexOf(".")+1));
68         className = className.substring(className.indexOf(".")+1);
69     }
70     if(!packageName.equalsIgnoreCase(" ")){
71         packageName=packageName.substring(0,packageName.length()-1);
72         doc.insertString(doc.getLength(), "package", reservedWordStyle);
73         doc.insertString(doc.getLength(), packageName+"\n\n", null);
74     }
75     Object[] imports = cc.getRefClasses().toArray();
76     for(int i=0;i<imports.length;i++){
77         String temp=(String)imports[i];
78         if(temp.indexOf("lang")==-1 && !temp.equalsIgnoreCase(cf.getName())){
79             doc.insertString(doc.getLength(), "import ", reservedWordStyle);
80             doc.insertString(doc.getLength(), temp+"\n", null);
81         }
82     }
83     if(imports.length > 0){
84         doc.insertString(doc.getLength(), "\n", null);
85     }
86     String accessFlags = Modifier.toString(cf.getAccessFlags());
87     int index = accessFlags.indexOf("synchronized");
88     if(index!=-1){
89         accessFlags = accessFlags.substring(0,index);
90     }
91     doc.insertString(doc.getLength(), accessFlags+" ", reservedWordStyle);
92     if(index!=-1){
93         doc.insertString(doc.getLength(), "class ", reservedWordStyle);
94     }
95     doc.insertString(doc.getLength(), className, null);
96     if(cf.getSuperclass()!=null && !cf.getSuperclass().equalsIgnoreCase("java.
97         lang.Object")){
98         doc.insertString(doc.getLength(), " extends ", reservedWordStyle);
99         doc.insertString(doc.getLength(), cf.getSuperclass(), null);
100    }
101    if(cf.getInterfaces().length!=0){
102        doc.insertString(doc.getLength(), " implements", reservedWordStyle);

```

```

102         for(int i=0;i<cf.getInterfaces().length;i++){
103             doc.insertString(doc.getLength(), " "+cf.getInterfaces()[i], null)
104             ;
105         }
106     }
107     doc.insertString(doc.getLength(), "{\n\n", null);
108     List list = cf.getFields();
109     int n = list.size();
110     for(int i=0; i<n; i++){
111         FieldInfo finfo = (FieldInfo)list.get(i);
112         int acc = finfo.getAccessFlags();
113         doc.insertString(doc.getLength(), " "+Modifier.toString(AccessFlag.
114             toModifier(acc)), reservedWordStyle);
115         String descriptor = finfo.getDescriptor();
116         int isArray = 0;
117         if(descriptor.indexOf('[')!=-1){
118             isArray=1;
119             descriptor = descriptor.substring(1);
120         }
121         if(dataTypes.get(descriptor)!=null){
122             doc.insertString(doc.getLength(), " " + dataTypes.get(descriptor),
123                 reservedWordStyle);
124         }
125         else{
126             int index2 = descriptor.indexOf("/");
127             while(index2!=-1){
128                 descriptor = descriptor.substring(index2+1);
129                 index2 = descriptor.indexOf("/");
130             }
131             if(descriptor.indexOf(";")!=-1){
132                 descriptor = descriptor.substring(0,descriptor.length()-1);
133             }
134             doc.insertString(doc.getLength(), " "+descriptor, reservedWordStyle
135                 );
136         }
137         if(isArray==1){
138             doc.insertString(doc.getLength(), "[ ]", null);
139         }
140         doc.insertString(doc.getLength(), " "+finfo.getName()+"\n", null);
141     }
142     doc.insertString(doc.getLength(), "\n", null);
143     CtConstructor[] constructors = cc.getDeclaredConstructors();
144     for(int i=0;i<constructors.length;i++){
145         MethodInfo minfo = constructors[i].getMethodInfo();
146         printMethod(cc, cp, minfo, doc, true, className);
147     }
148     list = cf.getMethods();

```

```

146         n = list.size();
147         for (int i = 0; i < n; ++i) {
148             MethodInfo minfo = (MethodInfo) list.get(i);
149             if (!minfo.getName().equalsIgnoreCase("<init>")) {
150                 printMethod(cc, cp, minfo, doc, false, className);
151             }
152         }
153         doc.insertString(doc.getLength(), "\n}", null);
154     }
155
156     public void disassembler(CtClass cc, MethodInfo minfo, Document doc) throws
157         Exception {
158         currentPC = 0;
159         currentVar = -1;
160         currentException = 0;
161         exceptionIndex = 0;
162         operandStack = new Stack();
163         branchNodeStack = new Stack<AbstractSyntaxTreeNode[]>();
164         gotoNodeStack = new Stack<AbstractSyntaxTreeNode[]>();
165         breakNodes = new HashMap<Integer, AbstractSyntaxTreeNode>();
166         currentInstruction = null;
167         previousInstruction = currentInstruction;
168         bytecode = null;
169         action = null;
170         symbol = null;
171         tree = new AbstractSyntaxTree(-1, null);
172         previousNode = tree.getRoot();
173         CodeAttribute code = minfo.getCodeAttribute();
174         ExceptionTable e = code.getExceptionTable();
175
176         ArrayList<Object []> sortedExceptions = this.exceptionTableSorting(e);
177
178         while (iterator.hasNext()) {
179             currentPC = (Integer) iterator.next();
180             currentInstruction = instructions.get(currentPC);
181             if (currentInstruction.indexOf(" ") != -1) {
182                 bytecode = currentInstruction.substring(0, currentInstruction.
183                     indexOf(" "));
184             }
185             else {
186                 bytecode = currentInstruction;
187             }
188
189             action = (SymbolTable.symbolTable.get(bytecode))[0];
190             symbol = (SymbolTable.symbolTable.get(bytecode))[1];
191
192             if (exceptionIndex < sortedExceptions.size()) {

```

```

192         exceptionHandling(sortedExceptions, e, cc);
193     }
194     if(!action.equalsIgnoreCase("noAction")){
195         Method method = JavaConversion.class.getDeclaredMethod(action+"
196             Action");
197         method.invoke(this);
198     }
199     previousInstruction = currentInstruction;
200     if(action.equalsIgnoreCase("desynchronize")){
201         continue;
202     }
203     System.out.println(currentPC + "\n");
204     iterator.remove();
205 }
206 tree.printTree(doc);
207
208 }
209
210 private ArrayList <Object []> exceptionTableSorting(ExceptionTable e){
211     ArrayList <Object []> sortedExceptions = new ArrayList <Object []>();
212     for(int i=0; i < e.size(); i++){
213         if(e.startPc(i) != e.handlerPc(i)){
214             sortedExceptions.add(new Object []{e.startPc(i), i});
215             sortedExceptions.add(new Object []{e.endPc(i), i});
216             sortedExceptions.add(new Object []{e.handlerPc(i), i});
217         }
218     }
219     bubbleSort(sortedExceptions);
220     for (int i=0; i<sortedExceptions.size(); i++){
221         System.out.println(sortedExceptions.get(i)[0]+" "+sortedExceptions.get(
222             i)[1]);
223     }
224     return sortedExceptions;
225 }
226
227 private void bubbleSort(ArrayList <Object []> sortedExceptions){
228     int j;
229     boolean flag = true;
230     Object [] temp;
231     while (flag) {
232         flag = false;
233         for(j = 0; j < sortedExceptions.size() - 1; j++){
234             if((Integer)sortedExceptions.get(j)[0] > (Integer)sortedExceptions.
235                 get(j+1)[0]){
236                 temp = sortedExceptions.get(j);
237                 sortedExceptions.set(j, sortedExceptions.get(j+1));
238                 sortedExceptions.set(j+1, temp);

```

```

238         flag = true;
239     }
240 }
241 }
242
243 private void exceptionHandling(ArrayList <Object []> sortedExceptions ,
244     ExceptionTable e, CtClass cc){
245     Object[] object = sortedExceptions.get(exceptionIndex);
246     int value = (Integer)object[0];
247     int index = (Integer)object[1];
248     int type = e.catchType(index);
249     if(value == e.startPc(index)){
250         // eliminating multiple starts in case there is an end or handle with
251         // the same value
252         int tempExceptionIndex = exceptionIndex + 1;
253         Object[] nextObject = sortedExceptions.get(tempExceptionIndex);
254         int nextValue = (Integer)nextObject[0];
255         int nextIndex = (Integer)nextObject[1];
256         while(value == nextValue){
257             if(nextValue != e.startPc(nextIndex)){
258                 object = nextObject;
259                 value = nextValue;
260                 index = nextIndex;
261                 exceptionIndex = tempExceptionIndex;
262                 break;
263             }
264             else{
265                 tempExceptionIndex++;
266                 nextObject = sortedExceptions.get(tempExceptionIndex);
267                 nextValue = (Integer)nextObject[0];
268                 nextIndex = (Integer)nextObject[1];
269             }
270         }
271     }
272     String currentExceptionType = cc.getClassFile().getConstPool().getClassInfo
273         (type);
274     if(currentPC == value){
275         if(value == e.startPc(index)){
276             AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(
277                 currentPC, "try{", null, null);
278             tree.addNode(branchNodeStack, previousNode, currentNode, currentPC)
279                 ;
280             previousNode = currentNode;
281             //checking for nested try blocks
282             do{
283                 exceptionIndex++;
284                 int nextIndex = (Integer)sortedExceptions.get(exceptionIndex)

```

```

280         [1];
        if (value == ((Integer)sortedExceptions.get(exceptionIndex)[0]).
            intValue() && e.endPc(index) != e.endPc(nextIndex) && e.
            catchType(nextIndex) != 0){
            currentNode = new AbstractSyntaxTreeNode(currentPC, "try{",
                null, null);
282         tree.addNode(branchNodeStack, previousNode, currentNode,
            currentPC);
            previousNode = currentNode;
284     }
    } while (value == ((Integer)sortedExceptions.get(exceptionIndex)[0]).
        intValue());
286 }
    else if (value == e.endPc(index)){
288         AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(
            currentPC, "}", null, null);
            tree.addNode(branchNodeStack, previousNode, currentNode, currentPC)
                ;
290         previousNode = currentNode;
            action = "noAction";
292         exceptionIndex++;
            // after an "end" there must be only a non-null "handle" with the
            same index or with a smaller startPC
294         Object[] nextObject = sortedExceptions.get(exceptionIndex);
            int nextValue = (Integer) nextObject[0];
296         int nextIndex = (Integer) nextObject[1];
            if (nextValue != e.handlerPc(nextIndex)){
298             while ((index != nextIndex && nextValue != e.handlerPc(index))
                && !(e.startPc(nextIndex) < e.startPc(index) && e.catchType
                    (nextIndex) != 0)){
                nextObject = sortedExceptions.get(++exceptionIndex);
                nextValue = (Integer) nextObject[0];
                nextIndex = (Integer) nextObject[1];
300             }
302         }
    }
304 }
    else if (value == e.handlerPc(index)){
306         AbstractSyntaxTreeNode currentNode = null;
            if (currentExceptionType != null){
308             currentExceptionType = currentExceptionType.substring(
                currentExceptionType.lastIndexOf(".") + 1);
                currentNode = new AbstractSyntaxTreeNode(currentPC, "catch(" +
                    currentExceptionType + " e" + (++currentException) + "){",
                    null, null);
310             if (exceptionIndex == sortedExceptions.size()-1 || (Integer)
                sortedExceptions.get(exceptionIndex+1)[0] != value){
                //AbstractSyntaxTreeNode endingNode = new
                AbstractSyntaxTreeNode(Integer.parseInt(instructions.

```

```

        get(idx - 1).toString().substring(5)), "}", null, null)
        ;
312     AbstractSyntaxTreeNode endingNode = new
        AbstractSyntaxTreeNode(Integer.parseInt(
            previousInstruction.substring(5)), "}", null, null);
        currentNode.setRight(endingNode);
314     }
    }
316     else{
        currentNode = new AbstractSyntaxTreeNode(currentPC, "finally{",
            null, null);
318         //AbstractSyntaxTreeNode endingNode = new
            AbstractSyntaxTreeNode(Integer.parseInt(instructions.get(
                idx - 1).toString().substring(5)), "}", null, null);
        AbstractSyntaxTreeNode endingNode = new AbstractSyntaxTreeNode(
            Integer.parseInt(previousInstruction.substring(5)), "}",
            null, null);
320         currentNode.setRight(endingNode);
    }
322     tree.addNode(branchNodeStack, previousNode, currentNode, currentPC)
        ;
        previousNode = currentNode;
324     action = "noAction";
        do{
326         exceptionIndex++;
    }while(exceptionIndex < sortedExceptions.size() && value == ((
        Integer)sortedExceptions.get(exceptionIndex)[0]).intValue());
328     }
    }
330     else if(currentPC < value){
        if(value == e.handlerPc(index)){
332         action = "noAction";
        }
    }
334     }
        else{
336         exceptionIndex++;
    }
338 }

340
342 private void pushAction(){
    //the symbol represents the value to be pushed
    if(symbol == null){
344         operandStack.push("var" + currentInstruction.substring(
            currentInstruction.indexOf(" ") + 1));
    }
346     else if(symbol.equalsIgnoreCase("["){ //the variable to be pushed is an
        array cell

```

```

348         String index = operandStack.pop().toString();
349         String arrayref = operandStack.pop().toString();
350         operandStack.push(arrayref + "[" + index + "]");
351     }
352     else if(symbol.equalsIgnoreCase("-")){ //bipush, sipush, iconst, lconst
353         case
354         operandStack.push(currentInstruction.substring(currentInstruction.
355             indexOf(" ") + 1));
356     }
357     else{ //symbol!=null
358         if(symbol.equalsIgnoreCase("this")){
359             if(varTypes.get(0)!=null){
360                 symbol = "var0";
361             }
362         }
363         operandStack.push(symbol);
364     }
365 }

366 private void popAction(){
367     AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(currentPC,
368         null, null);
369     currentNode.setData(operandStack.pop().toString() + ";");
370     tree.addNode(branchNodeStack, previousNode, currentNode, currentPC);
371     previousNode = currentNode;
372 }

373 private void operationAction(){
374     //the symbol represents the operation type
375     String op1 = operandStack.pop().toString();
376     String op2 = operandStack.pop().toString();
377     operandStack.push("(" + op1 + symbol + op2 + ")");
378 }

379 private void storeAction(){
380     //the symbol represents the variable type
381     String value = operandStack.pop().toString();
382     //AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(currentPC
383         , null, null);
384     //tree.addNode(branchNodeStack, previousNode, currentNode, currentPC);
385     //previousNode = currentNode;
386     if(symbol != null && (!symbol.equalsIgnoreCase("["))){
387         AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(
388             currentPC, null, null);
389         tree.addNode(branchNodeStack, previousNode, currentNode, currentPC);
390         previousNode = currentNode;
391         int index = Integer.parseInt(currentInstruction.substring(7));
392         if(currentVar < index){

```

```

390         currentVar = index;
391         currentNode.setData(symbol + " var" + index + " = " + value + ";");
392         varTypes.put(index, new Object[]{null, null});
393     }
394     else{
395         if(varTypes.get(index) != null){
396             currentNode.setData("var" + index + " = " + value + ";");
397         }
398         else{
399             currentNode.setData(symbol + " var" + index + " = " + value + "
400                                     ;");
401             varTypes.put(index, new Object[]{null, null});
402         }
403     }
404     }
405     else if(symbol != null){ //the variable is an array cell
406         String index = operandStack.pop().toString();
407         String arrayref = operandStack.pop().toString();
408         if(arrayref.startsWith("new")){
409             arrayref = arrayref.replaceAll("\\[[0-9]*\\]", "[ ]");
410             if(arrayref.contains("{")){
411                 operandStack.push(arrayref.substring(0, arrayref.length()-1) +
412                                     ", " + value + "}");
413             }
414             else{
415                 operandStack.push(arrayref + "{" + value + "}");
416             }
417         }
418         else{
419             AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(
420                 currentPC, null, null);
421             tree.addNode(branchNodeStack, previousNode, currentNode, currentPC)
422                 ;
423             currentNode.setData(arrayref + "[" + index + "] = " + value + ";");
424         }
425     }
426     }
427     else{ // astore case
428         if(value.contains("{new")){
429             value = value.replaceAll("\\{new [a-zA-Z]*[\\[ \\]]*", "{");
430             value = value.replaceAll(",new [a-zA-Z]*[\\[ \\]]*", ",");
431         }
432         AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(
433             currentPC, null, null);
434         tree.addNode(branchNodeStack, previousNode, currentNode, currentPC);
435         previousNode = currentNode;
436         int index = Integer.parseInt(currentInstruction.substring(7));
437         if(currentVar < index || varTypes.get(index) == null){
438             if(value.indexOf('') == 0){

```

```

434         varTypes.put(index, new Object[]{currentNode, "String"});
         currentNode.setData("String var" + index + " = " + value + ";")
         ;
    }
436    else if(value.equalsIgnoreCase("null")){
         varTypes.put(index, new Object[]{currentNode, "Object"});
438         currentNode.setData("Object var" + index + " = " + value + ";")
         ;
    }
440    else if(value.indexOf("new") == 0) {
         String type = value.toString();
442         if(type.indexOf("{") != -1){
             type = type.substring(type.indexOf(" ") + 1, type.indexOf("
{"));
444         }
         else if(type.indexOf("[") != -1){
446             String type2 = type.substring(type.indexOf(" ") + 1, type.
                indexOf("["));
             for(int i = 1; i < type.length(); i++){
448                 if(type.charAt(i) == ' '){
                     type2 = type2 + "[ ]";
450                 }
             }
452             type = type2;
         }
454         else if(type.indexOf("(") != -1){
             type = type.substring(type.indexOf(" ") + 1, type.indexOf("
("));
456         }
         currentNode.setData(type + " var" + index + " = " + value + ";"
            );
458         varTypes.put(index, new Object[]{currentNode, type});
    }
460    else if(value.contains("valueOf")){
         String type = value.substring(0, value.indexOf(' '));
462         varTypes.put(index, new Object[]{currentNode, type});
         currentNode.setData(type + " var" + index + " = " + value + ";"
            );
464    }
    else if (value.indexOf("var") == 0){
466         String index2 = value.substring(value.indexOf("r") + 1);
         String type = null;
468         type = (String)(varTypes.get(Integer.parseInt(index2))[1]);
         currentNode.setData(type + " var" + index + " = " + value + ";"
            );
470         varTypes.put(index, new Object[]{currentNode, type});
    }
472    currentVar = index;

```

```

474     }
475     else{
476         currentNode.setData("var" + index + " = " + value + ";");
477         AbstractSyntaxTreeNode temp = (AbstractSyntaxTreeNode)(varTypes.get
478             (index)[0]);
479         previousNode = currentNode;
480         if(value.indexOf('\"') == 0){
481             value = "String";
482         }
483         else if(value.indexOf("var") == 0){
484             String index2 = value.substring(value.indexOf("r") + 1);
485             value = (String)(varTypes.get(Integer.parseInt(index2))[1]);
486         }
487         else{
488             if(value.indexOf(" ") != -1){
489                 value = value.substring(value.indexOf(" ") + 1);
490             }
491             if(value.indexOf("(") != -1){
492                 value = value.substring(0, value.indexOf("("));
493             }
494         }
495         if(temp!=null && !value.equalsIgnoreCase("null")){
496             temp.setData(temp.getData().substring(temp.getData().indexOf("
497                 ")));
498             temp.setData(value + temp.getData());
499         }
500         if(!value.equalsIgnoreCase("null")){
501             varTypes.get(index)[1] = value;
502         }
503     }
504 }
505
506 private void castAction(){
507     //the symbol here represents the casting type
508     String num = operandStack.pop().toString();
509     operandStack.push("(" + symbol + ")" + num + ")");
510 }
511
512 private void newAction(){
513     String className = null;
514     if(currentInstruction.indexOf(".") != -1){
515         className = currentInstruction.substring(currentInstruction.lastIndexOf
516             (".") + 1);
517     }
518     else{
519         className = currentInstruction.substring(currentInstruction.lastIndexOf
520             (" ") + 1);

```

```

    }
518     operandStack.push("new " + className);
    }

520
    private void createAction(){
522         //the symbol represents whether the array is of a standard type or of a
            reference type
        String type = currentInstruction;
524         if(currentInstruction.contains(symbol)){
            type = currentInstruction.substring(currentInstruction.lastIndexOf(
                symbol) + 1);
526         }
        if(type.contains(";")){
528             type = type.substring(0, type.indexOf(";"));
        }
530         if(symbol.equalsIgnoreCase(".") && !currentInstruction.contains(symbol)){
            type = currentInstruction.substring(currentInstruction.lastIndexOf(" ")
                + 1);
532             if(type.startsWith("["){
                type = type.substring(type.lastIndexOf("[") + 1);
534                 if(dataTypes.containsKey(type)){
                    type = dataTypes.get(type);
536                 }
            }
538         }
        for(int i=0; i < currentInstruction.length(); i++){
540             if(currentInstruction.charAt(i) == '['){
                type = type + "[ ]";
542             }
        }
544         String length = operandStack.pop().toString();
        operandStack.push("new " + type + "[" + length + "]");
546     }

548     private void multiCreationAction(){
        String type = currentInstruction.substring(currentInstruction.lastIndexOf("
            [") + 1);
550         int dimensions = 0;
        String dims = currentInstruction.substring(currentInstruction.indexOf("[")
            );
552         for(int j = 0; j < dims.length(); j++){
            if (dims.charAt(j) == '['){
554                 dimensions++;
            }
556         }
        String formedDims = "";
558         for(int j = 0; j < dimensions; j++) {
            formedDims = "[" + operandStack.pop() + "]" + formedDims;

```

```

560     }
    String typeFormed = "";
562     if(dataTypes.get(type) != null){
        typeFormed = dataTypes.get(type);
564     }
    else{
566         typeFormed = type.substring(type.lastIndexOf(".") + 1, type.length() -
            1);
    }
568     operandStack.push("new " + typeFormed + formedDims);
}

570
private void swapAction(){
572     String op1 = operandStack.pop().toString();
    String op2 = operandStack.pop().toString();
574     operandStack.push(op1);
    operandStack.push(op2);
576 }

private void singleOperationAction(){
578     operandStack.push("~"+operandStack.pop().toString()+"");
580 }

private void incrementAction(){
582     int index = Integer.parseInt(currentInstruction.substring(
        currentInstruction.indexOf(" ") + 1, currentInstruction.lastIndexOf(" "
        )));
584     int value = Integer.parseInt(currentInstruction.substring(
        currentInstruction.lastIndexOf(" ") + 1));
    AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(currentPC,
        null, null);
586     tree.addNode(branchNodeStack, previousNode, currentNode, currentPC);
    previousNode = currentNode;
588     if(value < 128){
        currentNode.setData("var" + index + "+= " + value + ";");
590     }
    //else if(value < 256 && idx > 0 && !((String)instructions.get(idx-1)).
    equalsIgnoreCase("wide")){
592     else if(value < 256 && previousInstruction != null && !previousInstruction.
        equalsIgnoreCase("wide")){
        value = 128 - (value - 128);
594         currentNode.setData("var" + index + "-= " + value + ";");
    }
596     else if(value < 32768){
        currentNode.setData("var" + index + "+= " + value + ";");
598     }
    else{
600         value = 32768 - (value - 32768);

```

```

        currentNode.setData("var" + index + "-=" + value + ";");
    }
}

private void lengthAction(){
    String arrayref = operandStack.pop().toString();
    operandStack.push(arrayref + ".length");
}

private void instanceAction(){
    operandStack.push(operandStack.pop()+" instanceof class");
}

private void throwAction(){
    String throwValue = operandStack.pop().toString();
    throwValue = throwValue.substring(3);
    if(varTypes.containsKey(Integer.parseInt(throwValue))){
        AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(
            currentPC, null, null);
        currentNode.setData("throw var" + throwValue + ";");
        tree.addNode(branchNodeStack, previousNode, currentNode, currentPC);
    }
}

private void returnAction(){
    Object obj = operandStack.pop();
    AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(currentPC,
        null, null);
    currentNode.setData("return " + obj + ";");
    tree.addNode(branchNodeStack, previousNode, currentNode, currentPC);
    previousNode = currentNode;
}

private void getFieldAction(){
    String field = currentInstruction.substring(currentInstruction.lastIndexOf(
        ".") + 1);
    field = field.substring(0, field.indexOf("("));
    String ref = operandStack.pop().toString();
    operandStack.push(ref + "." + field);
}

private void putFieldAction(){
    String value = operandStack.pop().toString();
    String objectref = operandStack.pop().toString();
    String field = currentInstruction.substring(currentInstruction.lastIndexOf(
        ".") + 1, currentInstruction.indexOf("("));
    AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(currentPC,
        null, null);

```

```

644         tree.addNode(branchNodeStack, previousNode, currentNode, currentPC);
        previousNode = currentNode;
646         currentNode.setData(objectref + "." + field + " = " + value + ";");
    }
648
    private void putStaticAction() throws Exception{
650         String value = operandStack.pop().toString();
        String field = currentInstruction.substring(currentInstruction.lastIndexOf(
            ".") + 1, currentInstruction.indexOf("("));
652         String className = currentInstruction.substring(currentInstruction.
            lastIndexOf(" ") + 1, currentInstruction.lastIndexOf("."));
        ClassPool cp = new ClassPool();
654         cp.insertClassPath(m.getAction().getParentFilePath());
        CtClass cc = cp.get(m.getAction().getFileName());
656         ClassFile cf = cc.getClassFile();
        String classname = cf.getName();
658         AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(currentPC,
            null, null);
        tree.addNode(branchNodeStack, previousNode, currentNode, currentPC);
660         previousNode = currentNode;
        if(className.equalsIgnoreCase(classname)){
662             currentNode.setData(field + " = " + value + ";");
        }
664         else{
            currentNode.setData(className + "." + field + " = " + value + ";");
666         }
    }
668
    private void getStaticAction()throws Exception{
670         String field = currentInstruction.substring(currentInstruction.lastIndexOf(
            ".") + 1, currentInstruction.indexOf("("));
        String className = currentInstruction.substring(currentInstruction.
            lastIndexOf(" ") + 1, currentInstruction.lastIndexOf("."));
672         if(className.indexOf(".") != -1){
            className = className.substring(className.lastIndexOf(".") + 1);
674         }
        ClassPool cp = new ClassPool();
676         cp.insertClassPath(m.getAction().getParentFilePath());
        CtClass cc = cp.get(m.getAction().getFileName());
678         ClassFile cf = cc.getClassFile();
        String classname = cf.getName();
680         if(className.equalsIgnoreCase(classname)){
            operandStack.push(field);
682         }
        else{
684             operandStack.push(className + "." + field);
        }
686     }

```

```

688     private void checkcastAction(){
        String className = currentInstruction.substring(currentInstruction.
            lastIndexOf(".") + 1);
690     operandStack.push("(" + className + ")" + operandStack.pop());
    }
692
693     private void invokeAction(){
694         //the symbol represents the kind of method that gets invoked
        int numOfArgs = 0;
696         String args = null;
        if(currentInstruction.lastIndexOf("(") != currentInstruction.indexOf(")") -
            1){
698             args = currentInstruction.substring(currentInstruction.lastIndexOf("(")
                + 1, currentInstruction.indexOf(")"));
        }
700         if(args != null){
            int i = 0;
702             char previousChar = '\0';
            while (i < args.length()) {
704                 if(!symbol.equalsIgnoreCase("special") && (args.charAt(i) >= 65 &&
                    args.charAt(i) <= 90 && previousChar != '/')){
                    numOfArgs++;
706                 }
                else if(args.charAt(i) >= 65 && args.charAt(i) <= 90 &&
                    previousChar != '/' && (previousChar < 97 || previousChar >
                    122)){
708                     numOfArgs++;
                }
710                 previousChar = args.charAt(i);
                i++;
712             }
        }
714         if(currentInstruction.contains("util") && args != null && args.contains("
            Object")){ //to be fixed
            genericsHandling(numOfArgs);
716         }
        args = "";
718         for(int i = 0; i < numOfArgs; i++){
            args = operandStack.pop() + "," + args;
720         }
        if(!args.equalsIgnoreCase("")) {
722             args = args.substring(0, args.length() - 1);
        }
724         if(!symbol.equalsIgnoreCase("special")){
            String method = currentInstruction.substring(currentInstruction.
                lastIndexOf(".") + 1, currentInstruction.indexOf("("));
726         if(symbol.equalsIgnoreCase("virtual")){

```

```

728         method = operandStack.pop() + "." + method + "(" + args + ")";
    }
    else{
730         String className = currentInstruction.substring(0,
            currentInstruction.lastIndexOf("."));
        className = className.substring(className.lastIndexOf(".") + 1);
732         method = className + "." + method + "(" + args + ")";
    }
    String returnType = currentInstruction.substring(currentInstruction.
        indexOf("(") + 1, currentInstruction.lastIndexOf(")"));
734     if(returnType.equalsIgnoreCase("v")){
        AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(
736             currentPC, null, null);
        currentNode.setData(method + ";");
738         tree.addNode(branchNodeStack, previousNode, currentNode, currentPC)
            ;
        previousNode = currentNode;
    }
    else{
742         operandStack.push(method);
    }
744 }
    else{
746     String className = null;
        className = currentInstruction.substring(0, currentInstruction.indexOf(
            "<") - 1);
748     className = className.substring(className.lastIndexOf(".") + 1);
        String temp0 = operandStack.pop().toString();
750     if(temp0.equalsIgnoreCase("this")){
        AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(
            currentPC, null, null);
752         currentNode.setData("super(" + args + ");");
        tree.addNode(branchNodeStack, previousNode, currentNode, currentPC)
            ;
754         previousNode = currentNode;
    }
    else{
756         //String temp = (operandStack.pop() + "(" + args + ")");
        String temp = (temp0 + "(" + args + ")");
758         operandStack.push(temp);
    }
760 }
}

762

764 private void genericsHandling(int numOfArgs){
    String newType = "";
766     String delimiter = "";
    for(int i=1; i <= numOfArgs; i++){

```

```

768         String newArg = operandStack.elementAt(operandStack.size() - i).
            toString();
770         if(i > 1){
            delimiter = ", ";
772         }
774         if(newArg.startsWith("\\")){
            newType = "String" + delimiter + newType;
776         }
778         else if(newArg.contains(".valueOf")){
            newType = newArg.substring(0, newArg.indexOf('.') + 1) + delimiter +
                newType;
780         }
782         else if(newArg.startsWith("new")){
            newArg = newArg.substring(newArg.indexOf(' ') + 1) + delimiter +
                newType;
784         if(newArg.indexOf("(") != -1){
            newArg = newArg.substring(0, newArg.indexOf('('));
786         }
788         else if(newArg.indexOf("[") != -1){
            newArg = newArg.replaceAll("\\[[0-9]*\\]", "[ ]");
790         }
792         if(newArg.contains("{")){
            newArg = newArg.substring(0, newArg.indexOf('{'));
794         }
796         newType = newArg + delimiter + newType;
798         }
800         else if (newArg.startsWith("var")){
            newType = varTypes.get(Integer.parseInt(newArg.substring(3)))[1].
                toString() + delimiter + newType;
802         }
804         String [] newTypes = newType.split(",");
        for(int i = 0; i < newTypes.length; i++){
            newTypes[i] = newTypes[i].trim();
        }
        String variable = operandStack.elementAt(operandStack.size() - (numOfArgs +
            1)).toString();
        String varIndex = variable.substring(3);
        AbstractSyntaxTreeNode nodeToChange = (AbstractSyntaxTreeNode) varTypes.get
            (Integer.parseInt(varIndex))[0];
        String oldType = varTypes.get(Integer.parseInt(varIndex))[1].toString();
        if(!oldType.contains("<")){ //to be fixed
804         if(!oldType.contains(".") && !oldType.startsWith("int") && !oldType.
            startsWith("float") && !oldType.startsWith("char") && !oldType.
            startsWith("double") && !oldType.startsWith("byte") && !oldType.
            startsWith("long") && !oldType.startsWith("short") && !oldType.
            startsWith("boolean") && !oldType.startsWith("String")){
            nodeToChange.setData(nodeToChange.getData().replaceAll(varTypes.get

```

```

        (Integer.parseInt(varIndex))[1].toString(), varTypes.get(
            Integer.parseInt(varIndex))[1].toString() + "<" + newType + ">"
        ));
806         varTypes.get(Integer.parseInt(varIndex))[1] = varTypes.get(Integer.
            parseInt(varIndex))[1] + "<" + newType + ">";
    }
808 }
    else{
810         delimiter = "";
        newType = "";
812         oldType = oldType.substring(oldType.indexOf('<') + 1, oldType.indexOf('
            >'));
        String[] oldTypes = oldType.split(",");
814         for(int i = 0; i < oldTypes.length; i++){
            if(i > 0){
816                 delimiter = ", ";
            }
818             oldTypes[i] = oldTypes[i].trim();
            if(oldTypes[i].equalsIgnoreCase("Object") || !oldTypes[i].
                equalsIgnoreCase("Object") && !oldTypes[i].equalsIgnoreCase(
                    newTypes[i])){
820                 newTypes[i] = "Object";
            }
822             newType = newType + delimiter + newTypes[i];
        }
824         nodeToChange.setData(nodeToChange.getData().replace(oldType, newType));
        varTypes.get(Integer.parseInt(varIndex))[1] = varTypes.get(Integer.
            parseInt(varIndex))[1].toString().replace(oldType, newType);
826     }

828 }

830 private void ldcAction(){
832     String value = currentInstruction.substring(currentInstruction.indexOf("=")
        + 1).trim();
        value = escapeCharacters(value);
834     if(value.charAt(0) != 34){
        if(value.indexOf(" ") != -1){
836             value = value.substring(value.lastIndexOf(" ").trim());
        }
838     }
        operandStack.push(value);
840 }

842 protected static String escapeCharacters(String value){
        if(value.contains("\\\\")){
844             value = value.replaceAll("\\\\", "\\\\");

```

```

846     }
847     if (value.contains("\n")){
848         value = value.replaceAll("\\n", "\\\\n");
849     }
850     if (value.contains("\t")){
851         value = value.replaceAll("\\t", "\\\\t");
852     }
853     if (value.contains("\r")){
854         value = value.replaceAll("\\r", "\\\\r");
855     }
856     if (value.contains("\b")){
857         value = value.replaceAll("\\b", "\\\\b");
858     }
859     if (value.contains("\f")){
860         value = value.replaceAll("\\f", "\\\\f");
861     }
862     if (value.contains("\'")){
863         value = value.replaceAll("\\'", "\\\'");
864     }
865     /* if (value.indexOf("\") != -1 && value.indexOf("\'") != 0 && value.indexOf(
866         "\") != value.length()-1){
867         value = value.substring(1, value.length()-1);
868         value = value.replaceAll("\\"", "\\\\"");
869         value = "\"" + value + "\"";
870     } */
871     return value;
872 }
873
874 private void dupAction(){
875     //the symbol represents the kind of duplication
876     if (symbol == null){
877         //operandStack.push(operandStack.peek());
878     }
879     else if (symbol.equalsIgnoreCase("x1")){
880         operandStack.add(operandStack.size() - 3, operandStack.peek());
881     }
882     else if (symbol.equalsIgnoreCase("x2")){
883         operandStack.add(operandStack.size() - 4, operandStack.peek());
884     }
885     else if (symbol.equalsIgnoreCase("2")){
886         operandStack.push(operandStack.get(operandStack.size() - 2));
887         operandStack.push(operandStack.get(operandStack.size() - 2));
888     }
889     else if (symbol.equalsIgnoreCase("2_x1")){
890         operandStack.add(operandStack.size() - 4, operandStack.peek());
891         operandStack.add(operandStack.size() - 5, operandStack.get(operandStack
            .size() - 2));

```

```

    }
892     else if(symbol.equalsIgnoreCase("2_x2")){
            operandStack.add(operandStack.size() - 5, operandStack.peek());
894             operandStack.add(operandStack.size() - 6, operandStack.get(operandStack
                .size() - 2));
        }
896     }

898     private void synchronizeAction(){
        //using previousNode as currentNode because the previous node is an astore
        //node used internally for the monitorenter
900         currentVar -= 1;
        exceptionIndex++; //skipping try block involved in synchronization
902         previousNode.setData("synchronized(" + operandStack.pop() + "){");
    }

904     private void desynchronizeAction(){
906         exceptionIndex += 2; //skipping catch & finally blocks involved in
            synchronization
        operandStack.pop();
908         AbstractSyntaxTreeNode currentNode = new AbstractSyntaxTreeNode(currentPC,
            null, null);
        tree.addNode(branchNodeStack, previousNode, currentNode, currentPC);
910         previousNode = currentNode;
        currentNode.setData("}");
912         currentPC = (Integer)iterator.next();
        currentInstruction = (String) instructions.get(currentPC);
914         int gotoValue = Integer.parseInt(currentInstruction.substring(
            currentInstruction.indexOf(" ") + 1));
        iterator.remove();
916         while(currentPC < gotoValue - 1){ //discard all instructions until athrow
            currentPC = (Integer)iterator.next();
918             iterator.remove();
        }
920     }

922     private void branchAction(){
        if(symbol == null){ //fcmp, dcmp, lcmp bytecodes
924             iterator.remove();
            currentPC = (Integer) iterator.next();
926             currentInstruction = (String)instructions.get(currentPC);
            bytecode = currentInstruction.substring(0, currentInstruction.indexOf("
                "));
928             symbol = (SymbolTable.symbolTable.get(bytecode))[1];
            symbol = symbol.substring(1, symbol.lastIndexOf(" "));
930         }
        int branchValue = Integer.parseInt(currentInstruction.substring(
            currentInstruction.indexOf(" ") + 1));

```

```

932 AbstractSyntaxTreeNode currentNode = null;
    if((branchValue < currentPC) && !symbol.equalsIgnoreCase("goto")){ //do
        while case
934     StringBuilder logicalOperator = new StringBuilder();
        StringBuilder leftBracket = new StringBuilder();
936     StringBuilder rightBracket = new StringBuilder();
        currentNode = setCurrentDoWhileNode(branchValue, logicalOperator);
938     if(currentNode == null){
        currentNode = new AbstractSyntaxTreeNode(currentPC, null, null);
940         createLoopNode(currentNode, branchValue);
        tree.addNode(branchNodeStack, previousNode, currentNode, currentPC)
            ;
942     }
    else{
944         while(!rightBracketList.isEmpty() && rightBracketList.remove(
            Integer.valueOf(currentPC + 3))){
            rightBracket.append(")");
946         }
    }
948     previousNode = currentNode;
    if(symbol.contains(" ")){ //cases if ifeq, ifne, iflt, ifgt, ifge, ifle
        , ifnull, ifnonnull
950         setDoWhileNodeData(currentNode, leftBracket.toString() +
            operandStack.pop() + symbol + rightBracket.toString(),
            logicalOperator.toString());
952     }
    else{
954         String operand1 = operandStack.pop().toString();
        String operand2 = operandStack.pop().toString();
956         setDoWhileNodeData(currentNode, leftBracket.toString() + operand2 +
            " " + symbol + " " + operand1 + rightBracket.toString(),
            logicalOperator.toString());
        }
958 }
    else if((branchValue < currentPC) && symbol.equalsIgnoreCase("goto")){ //
        while case
960     AbstractSyntaxTreeNode temp = branchNodeStack.pop()[0];
        temp.setData(temp.getData().replace("if", "while"));
962     if(previousNode.getRight() != null && previousNode.getRight().
        getPcValue() < previousNode.getPcValue()){ //previous node is do
        while node
964         currentNode = previousNode.getLeft();
    }
    else{
966         currentNode = previousNode.getRight();
    }
968     currentNode.setRight(temp);

```

```

previousNode = temp; //going back to while node
970 }
else if(branchValue > currentPC && !symbol.equalsIgnoreCase("goto")){
972     StringBuilder logicalOperator = new StringBuilder();
974     StringBuilder leftBracket = new StringBuilder();
976     StringBuilder rightBracket = new StringBuilder();
978     currentNode = setCurrentIfNode(branchValue, logicalOperator,
leftBracket, rightBracket);
980     if(currentNode == null){
currentNode = new AbstractSyntaxTreeNode(currentPC, null, null);
982     AbstractSyntaxTreeNode endingBranchNode = new
AbstractSyntaxTreeNode(branchValue, "}", null, null);
984     branchNodeStack.push(new AbstractSyntaxTreeNode[] { currentNode,
endingBranchNode});
986     currentNode.setRight(endingBranchNode);
tree.addNode(branchNodeStack, previousNode, currentNode, currentPC)
;
988     previousNode = currentNode;
}
symbol = inverseRelationalOperator(symbol);
990 if(symbol.contains(" ")){
992     setIfNodeData(currentNode, leftBracket.toString() + operandStack.
pop() + symbol + rightBracket.toString(), logicalOperator.
toString());
}
994 else{
String operand1 = operandStack.pop().toString();
String operand2 = operandStack.pop().toString();
996 setIfNodeData(currentNode, leftBracket.toString() + operand2 + " "
+ symbol + " " + operand1 + rightBracket.toString(),
logicalOperator.toString());
998 }
}
1000 else{ //goto going forward
AbstractSyntaxTreeNode[] caseNode = null;
1002 while(caseNodeList.size() != 0 && caseNodeList.peek()[0].getPcValue() <
currentPC){
caseNode = caseNodeList.pop();
1004 }
if((caseNodeList.size() != 0 && caseNodeList.peek()[0].getPcValue() ==
currentPC + 3)
|| caseNode != null && caseNode[1].getPcValue() == currentPC +
3){
1006 if(branchValue != caseNode[1].getPcValue()){ //there is a default
case
1008 AbstractSyntaxTreeNode currentCase = caseNode[0];
while(currentCase.getLeft() != caseNode[1]){
1010 currentCase = currentCase.getLeft();

```

```

1006         }
        AbstractSyntaxTreeNode defaultCaseNode = new
            AbstractSyntaxTreeNode(caseNode[1].getPcValue(), null, null
            );
        defaultCaseNode.setData("default :{");
1008        caseNodeList.add(new AbstractSyntaxTreeNode[] {defaultCaseNode,
            caseNode[1]});
        AbstractSyntaxTreeNode endingDefaultCaseNode = new
            AbstractSyntaxTreeNode(branchValue, "}", null, null);
1010        defaultCaseNode.setRight(endingDefaultCaseNode);
        endingDefaultCaseNode.setRight(caseNode[1]);
1012        currentCase.setLeft(defaultCaseNode);
        if(currentCase != caseNode[0]){
1014            currentCase.getRight().setRight(defaultCaseNode);
        }
1016    }
    AbstractSyntaxTreeNode breakNode = new AbstractSyntaxTreeNode(
        currentPC, "break;", null, null);
1018    tree.addNode(branchNodeStack, previousNode, breakNode, currentPC);
    AbstractSyntaxTreeNode endingCaseNode = breakNode.getRight();
1020    endingCaseNode.setRight(caseNode[1]);
    endingCaseNode.getRight().setPcValue(branchValue);
1022    previousNode = caseNode[0];
}
1024 else{
    if(!gotoNodeStack.empty() && branchValue == gotoNodeStack.peek()
        [1].getPcValue()){//else if case
1026        AbstractSyntaxTreeNode previousElseNode = gotoNodeStack.peek()
            [0];
        AbstractSyntaxTreeNode ifNode = previousElseNode.getRight();
1028        previousElseNode.setData("else " + ifNode.getData());
        previousElseNode.setRight(ifNode.getRight());
        previousElseNode.setLeft(ifNode.getLeft().getRight());
1030        while(branchNodeStack.peek()[0] != ifNode){
            if(branchNodeStack.peek()[0].getLeft() != null){
1032                branchNodeStack.peek()[1].setRight(branchNodeStack.peek
                    () [0].getLeft());
            }
1034            branchNodeStack.pop();
        }
1036        branchNodeStack.peek()[1].setLeft(ifNode.getLeft().getLeft());
        ifNode.setLeft(null);
1038        ifNode.setRight(null);
        branchNodeStack.peek()[0] = previousElseNode;
        gotoNodeStack.pop();
1040    }
    AbstractSyntaxTreeNode endingBranchNode = new
        AbstractSyntaxTreeNode(branchValue, "}", null, null);
1042

```

```

1044         while(!branchNodeStack.empty() && branchNodeStack.peek()[1].
            getPcValue() <= branchValue){
            if(branchNodeStack.peek()[0].getLeft() != null){
1046                 branchNodeStack.peek()[1].setRight(branchNodeStack.peek()
                    [0].getLeft());
            }
1048             endingBranchNode.setLeft(branchNodeStack.peek()[1]);
            currentNode = branchNodeStack.pop()[0];
1050         }
        AbstractSyntaxTreeNode elseNode = new AbstractSyntaxTreeNode(
            endingBranchNode.getLeft().getPcValue(), "else{", null, null);
1052         elseNode.setRight(endingBranchNode);
        endingBranchNode.setRight(currentNode.getLeft());
1054         currentNode.setLeft(elseNode);
        previousNode = endingBranchNode;
1056         while(previousNode.getLeft() != null && previousNode.getLeft().
            getPcValue() < previousNode.getPcValue()){
            previousNode = previousNode.getLeft();
1058             previousNode.setRight(endingBranchNode.getRight());
        }
1060         previousNode = elseNode;
        gotoNodeStack.push(new AbstractSyntaxTreeNode[]{ elseNode,
            endingBranchNode});
1062     }
    }
1064 }

1066 private AbstractSyntaxTreeNode createLoopNode(AbstractSyntaxTreeNode
    currentNode, int branchValue){
    AbstractSyntaxTreeNode temp = tree.findNode(tree.getRoot(), branchValue);
1068     AbstractSyntaxTreeNode loopNode = new AbstractSyntaxTreeNode(branchValue,
        null, null, temp);
    AbstractSyntaxTreeNode whileNode = null;
1070     loopNode.setData("do{");
    if(temp == null){ //when the first statement is do...while
1072         tree.setRoot(loopNode);
    }
1074     else if(temp.getRight().getData().equalsIgnoreCase("do{") && (whileNode =
        findDoWhileNode(temp, currentNode)) != null){
        loopNode = temp.getRight();
1076         whileNode.setData(whileNode.getData().replace(";", " ||") +
            currentNode.getData().replace("}while(", " "));
        currentNode = whileNode;
1078         currentNode.setLeft(null);
    }
1080     else{
1082         loopNode.setRight(temp.getRight());
    }
}

```

```

1084         temp.setRight(loopNode);
1085     }
1086     currentNode.setRight(loopNode);
1087     return currentNode;
1088 }
1089
1090 private AbstractSyntaxTreeNode findDoWhileNode(AbstractSyntaxTreeNode temp,
1091     AbstractSyntaxTreeNode currentNode){
1092     AbstractSyntaxTreeNode current = temp.getRight();
1093     while(current.getRight() != temp.getRight()){
1094         if(current.getRight().getPcValue() < current.getPcValue()){
1095             current = current.getLeft();
1096         }
1097         else{
1098             current = current.getRight();
1099         }
1100     }
1101     if(current.getLeft() == currentNode){
1102         return current;
1103     }
1104     return null;
1105 }
1106
1107 private AbstractSyntaxTreeNode setCurrentDoWhileNode(int branchValue,
1108     StringBuilder logicalOperator){
1109     AbstractSyntaxTreeNode currentNode = null;
1110     String operator = "";
1111     if(previousNode != null && previousNode.getData().startsWith("}while") &&
1112         previousNode.getRight().getPcValue() == branchValue){
1113         currentNode = previousNode;
1114         operator = "||";
1115     }
1116     else if(previousNode.getData().startsWith("if") && previousNode.getRight().
1117         getPcValue() > branchValue){
1118         if(previousNode.getLeft() != null && previousNode.getLeft().getPcValue
1119             () > previousNode.getRight().getPcValue()){
1120             branchNodeConcatenation(branchValue);
1121         }
1122         operator = "&&";
1123         previousNode.setRight(null);
1124         previousNode.setData(previousNode.getData().replace("if", "}while").
1125             replace("{", ";"));
1126         currentNode = previousNode;
1127         currentNode = createLoopNode(currentNode, branchValue);
1128     }
1129     logicalOperator.append(operator);
1130     return currentNode;
1131 }

```

```

1126     }

    private void setDoWhileNodeData(AbstractSyntaxTreeNode currentNode, String data
        , String logicalOperator){
1128         if(currentNode.getData() == null){
            currentNode.setData("}while(" + data + ");");
1130         }
        else{
1132             currentNode.setData(currentNode.getData().replace(";", " " +
                logicalOperator + " " + data + ");"));
        }
1134     }

    private AbstractSyntaxTreeNode setCurrentIfNode(int branchValue, StringBuilder
        logicalOperator, StringBuilder leftBracket, StringBuilder rightBracket){
1136         AbstractSyntaxTreeNode currentNode = null;
1138         String operator = "";
        int previousBranchValue = -1;
1140         if(previousNode != null && previousNode.getData().startsWith("if") &&
            previousNode.getRight() != null){
            previousBranchValue = previousNode.getRight().getPcValue();
1142         }
        else if(previousNode == null || (previousNode != null && !previousNode.
            getData().startsWith("if"))){
1144             leftBracket.append("(");
            rightBracketList.add(branchValue);
1146             lowerPCValues.push(branchValue);
        }
1148         if(previousNode != null && previousNode.getData().startsWith("if") &&
            previousNode.getLeft() != null && previousNode.getLeft().getPcValue()
            <= branchValue && !containsGotoStatement(branchValue, previousNode.
            getLeft().getPcValue())){
            branchNodeConcatenation(branchValue);
1150         }
        if(previousNode != null && previousNode.getData().startsWith("if") &&
            previousBranchValue == branchValue){
1152             currentNode = previousNode;
            operator = "&&";
1154         }
        else if(previousNode != null && previousNode.getData().startsWith("if") &&
            (previousBranchValue == currentPC + 3 || (previousBranchValue <
            branchValue && previousBranchValue > currentPC + 3)) && !
            containsGotoStatement(branchValue, previousBranchValue)){
1156             currentNode = previousNode;
            currentNode.getRight().setPcValue(branchValue);
1158             operator = "||";
        }
1160         logicalOperator.append(operator);

```

```

1162         if(branchValue < previousBranchValue){
            if(!rightBracketList.isEmpty() && !rightBracketList.contains(
                branchValue)){
                leftBracket.append("(");
1164                lowerPCValues.push(branchValue);
            }
1166            else{
                leftBracket.append("(");
1168            }
            rightBracketList.add(branchValue);
1170        }
        while(!lowerPCValues.empty() && lowerPCValues.peek() < branchValue){
1172            rightBracketList.add(branchValue);
            lowerPCValues.pop();
1174        }
        while(!rightBracketList.isEmpty() && rightBracketList.remove(Integer.
            valueOf(currentPC + 3))){
1176            rightBracket.append(")");
        }
1178        return currentNode;
1180    }

1182    private void branchNodeConcatenation(int branchValue){
        AbstractSyntaxTreeNode[] branchNode2 = branchNodeStack.pop();
1184        AbstractSyntaxTreeNode[] branchNode1 = branchNodeStack.pop();
        String operator = "&&";
1186        if(previousNode.getLeft().getPcValue() == branchValue){
            operator = "||";
1188        }
        String data = branchNode2[0].getData().replace("if(", "").replace("){", "")
            ;
1190        setIfNodeData(branchNode1[0], data, operator);
        branchNode1[0].setRight(branchNode1[1]);
1192        branchNodeStack.push(branchNode1);
        previousNode = branchNode1[0];
1194    }

1196    private boolean containsGotoStatement(int branchValue, int previousBranchValue)
    {
        Iterator it = instructions.keySet().iterator();
1198        int key = (Integer)it.next();
        while(key < branchValue){
1200            if(instructions.get(key).startsWith("goto " + branchValue) && (
                previousBranchValue == key + 3 || previousBranchValue ==
                branchValue)){
                return true;
1202            }
        }
    }

```

```

1204         key = (Integer)it.next();
        }
        return false;
1206     }

1208     private String inverseRelationalOperator(String operator){
1209         if (operator.contains("==")){
1210             operator = operator.replace("==", "!=");
1211         }
1212         else if (operator.contains("!=")){
1213             operator = operator.replace("!=", "==");
1214         }
1215         else if (operator.contains("<=")){
1216             operator = operator.replace("<=", ">");
1217         }
1218         else if (operator.contains(">=")){
1219             operator = operator.replace(">=", "<");
1220         }
1221         else if (operator.contains("<")){
1222             operator = operator.replace("<", ">=");
1223         }
1224         else if (operator.contains(">")){
1225             operator = operator.replace(">", "<=");
1226         }
1227         else if (operator.contains("&")){
1228             operator = operator.replace("&", "|");
1229         }
1230         else if (operator.contains("|")){
1231             operator = operator.replace("|", "&");
1232         }
1233         else{
1234             operator = null;
1235         }
1236         return operator;
        }

1238     private void setIfNodeData(AbstractSyntaxTreeNode currentNode, String data,
        String logicalOperator){
1239         if(currentNode.getData() == null){
1240             currentNode.setData("if(" + data + "){"");
1241         }
1242         else{
1243             if(logicalOperator.equalsIgnoreCase("||")){
1244                 char[] previousData = currentNode.getData().toCharArray();
1245                 ArrayList<Character> newData = new ArrayList<Character>();
1246                 int i = 0;
1247                 for(i = 0; i < previousData.length-1; i++){
1248                     if(previousData[i] == ' '){

```

```

1250         newData.add(previousData[i]);
1252         continue;
1253     }
1254     String charsToCheck = String.valueOf(new char [] {
1255         previousData[i], previousData[i+1]});
1256     String checkResult = inverseRelationalOperator(charsToCheck);
1257     if(checkResult != null){
1258         newData.add(checkResult.charAt(0));
1259         if(checkResult.length() == 2){
1260             newData.add(checkResult.charAt(1));
1261         }
1262         i++;
1263     }
1264     else{
1265         newData.add(previousData[i]);
1266     }
1267 }
1268 newData.add(previousData[i]);
1269 currentNode.setData("");
1270 for(i = 0; i < newData.size(); i++){
1271     currentNode.setData(currentNode.getData() + newData.get(i));
1272 }
1273 currentNode.setData(currentNode.getData().replace("{}"," " +
1274     logicalOperator + " " + data + "{}"));
1275 }
1276 }
1277
1278 private void switchAction(){
1279     AbstractSyntaxTreeNode switchNode = new AbstractSyntaxTreeNode(currentPC,
1280         null, null);
1281     AbstractSyntaxTreeNode endSwitchNode = new AbstractSyntaxTreeNode(-1, null,
1282         null);
1283     endSwitchNode.setData("}aaa");
1284     AbstractSyntaxTreeNode currentNode = switchNode;
1285     currentNode.setData("switch(" + operandStack.pop() + ")");
1286     String[] cases = currentInstruction.split("\n");
1287     if(cases[1].contains("default")){
1288         String defaultCase = cases[1];
1289         for(int i = 1; i < cases.length - 1; i++){
1290             cases[i] = cases[i + 1];
1291         }
1292         cases[cases.length - 1] = defaultCase;
1293     }
1294     for(int i = 1; i < cases.length; i++){
1295         cases[i] = cases[i].substring(cases[i].lastIndexOf('\t') + 1);
1296         String[] values = cases[i].split(":");
1297         values[1] = values[1].substring(1);

```

```

1294         if(values[1].contains("}")){
1295             values[1] = values[1].substring(0, values[1].length() - 1);
1296         }
1297         if(Integer.parseInt(values[1]) == currentNode.getPcValue()){
1298             currentNode.setData(currentNode.getData().replace('{', ' '));
1299             currentNode.setData(currentNode.getData() + "case " + values[0] + "
1300                 :{");
1301         }
1302         else{
1303             if(cases[i].startsWith("default")){
1304                 currentNode.getRight().setPcValue(Integer.parseInt(values[1]));
1305                 currentNode.getRight().setRight(endSwitchNode);
1306                 endSwitchNode.setPcValue(Integer.parseInt(values[1]));
1307                 currentNode.setLeft(endSwitchNode);
1308             }
1309             else{
1310                 AbstractSyntaxTreeNode newCaseNode = new AbstractSyntaxTreeNode
1311                     (Integer.parseInt(values[1]), null, null);
1312                 newCaseNode.setData("case " + values[0] + ":{");
1313                 AbstractSyntaxTreeNode endingCaseNode = new
1314                     AbstractSyntaxTreeNode(-1, "}", null, null);
1315                 newCaseNode.setRight(endingCaseNode);
1316                 caseNodeList.add(new AbstractSyntaxTreeNode[] {newCaseNode,
1317                     endSwitchNode});
1318                 if(currentNode.getData().startsWith("case")){
1319                     currentNode.getRight().setPcValue(newCaseNode.getPcValue())
1320                         ;
1321                     currentNode.setLeft(newCaseNode);
1322                     currentNode.getRight().setRight(newCaseNode);
1323                 }
1324                 else{
1325                     currentNode.setRight(newCaseNode);
1326                 }
1327                 currentNode = newCaseNode;
1328             }
1329         }
1330     }
1331     tree.addNode(branchNodeStack, previousNode, switchNode, currentPC);
1332     previousNode = switchNode;
1333 }
1334
1335 public void printMethod(CtClass cc, ClassPool cp, MethodInfo minfo, Document
1336     doc, Boolean isConstructor, String className) throws Exception{
1337     ExceptionsAttribute exa = minfo.getExceptionsAttribute();
1338     CodeAttribute code = minfo.getCodeAttribute();
1339     ExceptionTable e = code.getExceptionTable();
1340     for(int i =0; i<e.size(); i++){

```

```

1336         System.out.println(e.startPc(i)+"start");
1337         System.out.println(e.endPc(i)+"end");
1338         System.out.println(e.handlerPc(i)+"handler");
1339         System.out.println(cc.getClassFile().getConstPool().getClassInfo(e.
            catchType(i)));
1340     }
1341     String[] exceptions = null;
1342     if(exa!=null){
1343         exceptions = minfo.getExceptionsAttribute().getExceptions();
1344     }
1345     int acc = minfo.getAccessFlags();
1346     CtClass returnTypeClass = Descriptor.getReturnType(minfo.getDescriptor(),cp
        );
1347     String returnType = minfo.getDescriptor();
1348     if(returnTypeClass != null){
1349         returnType = Descriptor.getReturnType(minfo.getDescriptor(),cp).
            toString();
1350     }
1351     else{
1352         returnType = returnType.substring(returnType.lastIndexOf('/')+1,
            returnType.length()-1);
1353     }
1354     if(returnType != null && returnType.indexOf('[') != -1){
1355         returnType = returnType.substring(returnType.indexOf("[")+1,returnType.
            indexOf("]"));
1356     }
1357     if(isConstructor){
1358         doc.insertString(doc.getLength(), "    "+Modifier.toString(AccessFlag.
            toModifier(acc)), reservedWordStyle);
1359     }
1360     else{
1361         doc.insertString(doc.getLength(), "    "+Modifier.toString(AccessFlag.
            toModifier(acc))+ " "+ returnType+" ", reservedWordStyle);
1362     }
1363     if(isConstructor){
1364         doc.insertString(doc.getLength(), " " + className + "(" , null);
1365     }
1366     else{
1367         doc.insertString(doc.getLength(), minfo.getName()+"(" , null);
1368     }
1369     String arguments = minfo.getDescriptor().substring(1,minfo.getDescriptor().
        indexOf(")"));
1370     int i=0;
1371     int varCount = 1;
1372     if(Modifier.toString(AccessFlag.toModifier(acc)).contains("static")){ //
        static methods don't use this so astore_0 refers to var0
        varCount = 0;
1373     }
1374 }

```

```

1376     int array = 0;
1377     while(i<arguments.length()){
1378         int firstCase = 0;
1379         int standardType = 0;
1380         if(i!= arguments.length()-1 && arguments.charAt(i+1)>=97 && arguments.
            charAt(i+1)<=122){
1381             String newArg = minfo.getDescriptor().substring(i);
1382             newArg = newArg.substring(0, newArg.indexOf(";"));
1383             i+=newArg.length();
1384             if(array != 1){
1385                 doc.insertString(doc.getLength(), newArg.substring(newArg.
                    lastIndexOf("/")+1)+" var"+varCount, null);
1386             }
1387             else{
1388                 doc.insertString(doc.getLength(), newArg.substring(newArg.
                    lastIndexOf("/")+1)+"[ ] var"+varCount, null);
1389                 array = 0;
1390             }
1391             varTypes.put(varCount, new Object [] { null, newArg.substring(newArg.
                lastIndexOf("/")+1)});
1392             firstCase = 1;
1393         }
1394         else if(arguments.charAt(i)=='['){
1395             array = 1;
1396             i++;
1397             continue;
1398         }
1399         else{
1400             if(array!=1){
1401                 doc.insertString(doc.getLength(), dataTypes.get(String.valueOf(
                    arguments.charAt(i))+" ", reservedWordStyle);
1402             }
1403             else{
1404                 doc.insertString(doc.getLength(), dataTypes.get(String.valueOf(
                    arguments.charAt(i))+"[ ] ", reservedWordStyle);
1405             }
1406             doc.insertString(doc.getLength(), "var"+varCount, null);
1407             varTypes.put(varCount, new Object [] { null, null});
1408             standardType = 1;
1409         }
1410         if((i<=arguments.length()-1 && standardType == 0) || (standardType == 1
            && i<arguments.length()-1)){
1411             doc.insertString(doc.getLength(), ", ", null);
1412         }
1413         if(firstCase == 0){
1414             i++;
1415         }
1416         varCount++;

```

```

1416         standardType = 0;
1417     }
1418     doc.insertString(doc.getLength(), ")", null);
1419     if(exceptions!=null){
1420         doc.insertString(doc.getLength(), " throws ", reservedWordStyle);
1421         for(int j=0;j<exceptions.length-1;j++){
1422             doc.insertString(doc.getLength(), exceptions[j].substring(
                exceptions[j].lastIndexOf(".")+1)+", ", null);
1423         }
1424         doc.insertString(doc.getLength(), exceptions[exceptions.length-1].
            substring(exceptions[exceptions.length-1].lastIndexOf(".")+1)+"() {\n
            n", null);
1425     }
1426     else{
1427         doc.insertString(doc.getLength(), "{\n", null);
1428     }
1429     CodeAttribute ca = minfo.getCodeAttribute();
1430     ConstPool pool = minfo.getConstPool();
1431     CodeIterator ci = ca.iterator();
1432     currentPC = 0;
1433     while(ci.hasNext()){
1434         int index2 = ci.next();
1435         String instruction = InstructionPrinter.instructionString(ci, index2,
            pool);
1436         if(instruction.startsWith("wide")){
1437             instruction = instruction.substring(5);
1438             instructions.put(currentPC, "wide");
1439             currentPC = currentPC + 3;
1440         }
1441         instructions.put(currentPC, instruction);
1442         if(instruction.indexOf(" ")!=-1){
1443             bytecode = instruction.substring(0, instruction.indexOf(" "));
1444         }
1445         else{
1446             bytecode = instruction;
1447         }
1448         currentPC = currentPC + calculatePC(bytecode, instruction, currentPC);
1449     }
1450     iterator = instructions.keySet().iterator();
1451     currentPC = 0;
1452     disassembler(cc, minfo, doc);
1453     doc.insertString(doc.getLength(), "    }\n", null);
1454 }

1455
1456 public static int calculatePC(String bytecode, String currentInstruction, int
    currentPC){
    if(bytecode.equalsIgnoreCase("astore")||bytecode.equalsIgnoreCase("bipush")
        || bytecode.equalsIgnoreCase("dload")){

```

```

1458         return 2;
    }
1460     if (bytecode.equalsIgnoreCase("dstore") || bytecode.equalsIgnoreCase("fload")
        || bytecode.equalsIgnoreCase("fstore")){
        return 2;
1462     }
    if (bytecode.equalsIgnoreCase("iload") || bytecode.equalsIgnoreCase("istore")
        || bytecode.equalsIgnoreCase("ldc") || bytecode.equalsIgnoreCase("lload")
        ){
1464         return 2;
    }
    if (bytecode.equalsIgnoreCase("lstore") || bytecode.equalsIgnoreCase("aload")
        || bytecode.equalsIgnoreCase("newarray") || bytecode.equalsIgnoreCase("ret")){
1466         return 2;
    }
    if (bytecode.equalsIgnoreCase("checkcast") || bytecode.equalsIgnoreCase("getfield") || bytecode.equalsIgnoreCase("getstatic") || bytecode.equalsIgnoreCase("goto")){
1468         return 3;
    }
    if (bytecode.equalsIgnoreCase("if_acmpeq") || bytecode.equalsIgnoreCase("if_acmpne") || bytecode.equalsIgnoreCase("if_icmpeq") || bytecode.equalsIgnoreCase("if_icmpne")){
1472         return 3;
    }
    if (bytecode.equalsIgnoreCase("if_icmplt") || bytecode.equalsIgnoreCase("if_icmpge") || bytecode.equalsIgnoreCase("if_icmpgt") || bytecode.equalsIgnoreCase("if_icmple")){
1474         return 3;
    }
    if (bytecode.equalsIgnoreCase("ifeq") || bytecode.equalsIgnoreCase("ifne") || bytecode.equalsIgnoreCase("iflt") || bytecode.equalsIgnoreCase("ifge")){
1476         return 3;
    }
    if (bytecode.equalsIgnoreCase("ifgt") || bytecode.equalsIgnoreCase("ifle") || bytecode.equalsIgnoreCase("ifnonnull") || bytecode.equalsIgnoreCase("ifnull")){
1480         return 3;
    }
    if (bytecode.equalsIgnoreCase("iinc") || bytecode.equalsIgnoreCase("instanceof") || bytecode.equalsIgnoreCase("invokespecial") || bytecode.equalsIgnoreCase("invokestatic")){
1482         return 3;
    }
    if (bytecode.equalsIgnoreCase("invokevirtual") || bytecode.equalsIgnoreCase("jsr") || bytecode.equalsIgnoreCase("ldc_w") || bytecode.equalsIgnoreCase("ldc2_w")){
1484
1486

```

```

1488         return 3;
1489     }
1490     if(bytecode.equalsIgnoreCase("anewarray") || bytecode.equalsIgnoreCase("new")
        || bytecode.equalsIgnoreCase("putfield") || bytecode.equalsIgnoreCase("
        putstatic") || bytecode.equalsIgnoreCase("sipush")){
1491         return 3;
1492     }
1493     if(bytecode.equalsIgnoreCase("multianewarray")){
1494         return 4;
1495     }
1496     if(bytecode.equalsIgnoreCase("wide")){
1497         return 3;
1498     }
1499     if(bytecode.equalsIgnoreCase("goto_w") || bytecode.equalsIgnoreCase("
        invokeinterface") || bytecode.equalsIgnoreCase("jsr_w")){
1500         return 5;
1501     }
1502     if(bytecode.equalsIgnoreCase("lookupswitch") || bytecode.equalsIgnoreCase("
        tableswitch")){
        String cases = currentInstruction.substring(currentInstruction.indexOf(
            "{")+1,currentInstruction.indexOf("}")).trim().replace('\n', ' ').
            replace('\t', ' ');
1503     String returnValue = null;
1504     if(cases.indexOf("default")!=0){
1505         returnValue = cases.substring(cases.indexOf(" ") +1);
1506         returnValue = returnValue.substring(0,returnValue.indexOf(" "));
1507     }
1508     else{
1509         returnValue = cases.substring(cases.indexOf(" ") +1);
1510         returnValue = returnValue.substring(returnValue.indexOf(":")+1);
1511         returnValue = returnValue.substring(returnValue.indexOf(" ") +1);
1512         returnValue = returnValue.substring(0,returnValue.indexOf(" "));
1513     }
1514     return Integer.parseInt(returnValue) - currentPC;
1515 }
1516 else{
1517     return 1;
1518 }
1519 }
1520 }
1521 }
1522 }

```

Βιβλιογραφία

- [1] H.M. Deitel, P.J. Deitel, *JAVA How to program*, 6th edition, Pearson Education, 2005.
- [2] Tim Lindholm, Frank Yellin , *The Java Virtual Machine Specification*, 2nd edition, Prentice Hall, 1999.
- [3] Tim Lindholm, Frank Yellin , *VM Spec: The Java Virtual Machine Instruction Set*, 2nd edition, Prentice Hall, 1999.
- [4] Jerome Miecznikowski , *New algorithms for a Java decompiler and their implementation in Soot*, McGill University, 2003.
- [5] <http://jode.sourceforge.net/>
- [6] [http://en.wikipedia.org/wiki/JAD_\(JAvA_Decompiler\)](http://en.wikipedia.org/wiki/JAD_(JAvA_Decompiler))
- [7] <http://java-decompiler.github.io/>
- [8] Dave Dyer, *Java decompilers compared*, JavaWorld.com, 1997.
- [9] <http://www.program-transformation.org/Transform/JavaDecompilers>
- [10] Shigeru Chiba, Muga Nishizawa, *An Easy-to-Use Toolkit for Efficient Java Bytecode Translators*, Proc. of 2nd Int'l Conf. on Generative Programming and Component Engineering (GPCE '03), LNCS 2830, pp.364-376, Springer-Verlag, 2003.
- [11] *Java 2 Platform Standard Edition 5.0 API Specification*
- [12] *Javassist API* (<http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/html/index.html>)
- [13] <https://www.benf.org/other/cfr/>

- [14] <https://github.com/mstrobel/procyon/wiki/Java-Decompiler>
- [15] <https://github.com/fesh0r/fernflower>
- [16] <https://medium.com/@vansh.rahangdale211/security-and-decompilation-the-role-of-decompilers-in-the-field-of-cybersecurity-and-malware-556991d78422>