ΜΕΤΑΓΛΩΤΤΙΣΤΕΣ

ΕΑΡΙΝΟ ΕΞΑΜΗΝΟ 2020

2Η ΕΡΓΑΣΙΑ

ΒΙΡΒΙΔΑΚΗ ΧΡΥΣΟΥΛΑ

ΑΜ: 1115201600021

Εντολή μεταγλώττισης: make

Εντολή εκτέλεσης: java Main [file1] [file2] ... [fileN]

ΔΟΜΕΣ

Για την εκπόνηση της εργασίας ήταν απαραίτητη η αποθήκευση των ονομάτων των κλάσεων, των συναρτήσεων και των μεταβλητών του κάθε αρχείου που δινόταν ως είσοδος στο πρόγραμμα. Πρώτο βήμα, λοιπόν , είναι η ανάλυση των δομών που χρησιμοποίησα για να αποθηκεύσω την πληροφορία μου (στον πρώτο Visitor) και να την οργανώσω με τέτοιο τρόπο ώστε να μου φανεί στη συνέχεια χρήσιμη (δεύτερος Visitor). Έφτιαξα, λοιπόν, μία λίστα που ονόμασα SymbolTable η οποία περιείχε όλη την πληροφορία οργανωμένη σε εσωτερικές λίστες. Η κάθε εσωτερική λίστα αποτελούσε και μία αλυσίδα από κλάσεις οι οποίες ήταν η μία υπερκλάση της άλλης. Κόμβοι αυτής της αλυσίδας ήταν HashMaps που συμβόλιζαν κάποια εσωτερικά scopes που δημιουργούνταν για παράδειγμα στο εσωτερικό μίας συνάρτησης μίας κλάσης. Το κάθε στοιχείο ενός HashMap ήταν ουσιαστικά μία αντιστοίχηση μίας μεταβλητής με τον τύπο της. Ο απλούστερος τρόπος να γίνει κατανοητό αυτό είναι ένα παράδειγμα. Έστω το μικρό πρόγραμμα:

```
class MainClass{
        public static void main(String[] a){
                int[] array;
                System.out.println(array.length);
        }
```

```
}
class A {
        int num;
        boolean bool;
        public int fun(){          int x; return x; }
}
class B extends A{
        int[] numB;
        public int[] boo(){       int x;    return numB; }
        public boolean blaa(){           int y;    return true; }
}
```

Έχουμε, λοιπόν, την κύρια κλάση και δύο ακόμα κλάσεις. Σκοπός μου με τις δομές που χρησιμοποίησα ήταν να μπορώ να αναζητώ κάποια συγκεκριμένη μεταβλητή εύκολα μέσα σε όλα τα δυνατά scopes τα οποία ήταν ορατά από ένα δεδομένο σημείο του προγράμματος. Αν δηλαδή βρισκόμουν μέσα σε μία συνάρτηση θα έπρεπε για να βρω μια μεταβλητή να ψάξω αρχικά μέσα στις μεταβλητές της συνάρτησης αυτής. Έπειτα αν δεν την έβρισκα εκεί θα έψαχνα στα πεδία της κλάσης στην οποία ανήκει η συνάρτηση. Κι αν ούτε τότε την έβρισκα θα έπρεπε να ψάξω στα πεδία της υπερκλάσης της κλάσης που άνηκε η συνάρτηση, αν φυσικά είχε, και ούτω κάθε εξής. Άρα για κάθε μία ξεχωριστή συνάρτηση μίας κλάσης έφτιαχνα την αλυσίδα των ορατών scopes. Το παραπάνω παράδειγμα, λοιπόν, θα γίνει:

```
[
 [{MainClass={a=String[], array=int[]} } ],
 [{A={num=int, bool=boolean}}, {fun={return_value=int, x=int} } ],
 [{B={numB=int[]}}, {boo={return_value=int[], x=int}}, {A={num=int, bool=boolean} } ],
[{B={numB=int[]}}, {blaa={return_value=boolean, y=int}},{A={num=int, bool=boolean} } ]
]
```

Επίσης για κάθε συνάρτηση αποθηκεύουμε ποιος είναι ο τύπος επιστροφής της.

Ακόμα αποθηκεύω για βοηθητικούς λόγους σε ένα HashMap τα ζευγάρια των υπερκλάσεων που υπάρχουν, δηλαδή στο προηγούμενο παράδειγμα μέσα στο HashMap

θα αποθηκευόταν το { B = A }. Αυτό με βοηθάει ώστε να βρίσκω εύκολα αν η κλάση που με ενδιαφέρει εκείνη τη στιγμή έχει κάποια υπερκλάση και ποιά.

Κατά την διάρκεια της αποθήκευσης των μεταβλητών υπολογίζω και τα offsets αυτών τα οποία και αποθηκεύω και πάλι σε HashMap ξεχωριστά για την κάθε κλάση. Το αποτέλεσμα του παραπάνω παραδείγματος για τα offsets είναι:

A.num: 0

A.bool: 4

A.fun: 0

B.numB: 5

B.boo: 8

B.blaa: 16

Έλεγχοι Λαθών

Στον πρώτο Visitor γίνεται μικρό τμήμα των ελέγχων. Για παράδειγμα σε αυτόν ελέγχω αν στο ίδιο scope δηλωθεί κάποια μεταβλητή δύο φορές είτε με τον ίδιο τύπο είτε με διαφορετικό. Ακόμα ελέγχω να μην δηλωθούν δύο φορές ονόματα κλάσεων και να μην δηλωθεί κάποια κλάση ως υποκλάση κάποιας άλλης η οποία ακόμα δεν έχει δηλωθεί. Επιπλέον προσέχω να μην υπάρξει δήλωση ενός ονόματος συνάρτησης και σε υπερκλάση και σε υποκλάση με διαφορετικό τύπο επιστροφής ή διαφορετικά (στο πλήθος ή στο είδος) ορίσματα. Αν μια συνάρτηση υπάρχει και σε υπερκλάση και σε υποκλάση θα πρέπει να είναι ακριβώς η ίδια.

Στον δεύτερο Visitor είναι που γίνονται κατά κύριο λόγο οι έλεγχοι τύπων. Χρησιμοποιώ μία global μεταβλητή type1 η οποία συμβολίζει ουσιαστικά των αναμενόμενο επιθυμητό τύπο σε κάθε περίπτωση. Για παράδειγμα η έκφραση μέσα στις παρενθέσεις ενός if θα αναμένεται να είναι τύπου boolean. Στην μεταβλητή type1 θα αποθηκευτεί ο τύπος που πραγματικά είναι η έκφραση μέσα στις παρενθέσεις του if και έπειτα θα ελεγχθεί αν είναι και ο επιθυμητός. Με το ίδιο τρόπο ελέγχονται κι άλλες παρόμοιες περιπτώσεις.

Αν διαπιστωθεί κάποιο λάθος τόσο στον πρώτο Visitor όσο και στον δεύτερο τότε φροντίζω να σταματήσει η διαδικασία. Στο τέλος του δεύτερου Visitor κι εφόσον δεν έχει εντοπιστεί κάποιο σφάλμα εκτυπώνω τα offsets. Σε περίπτωση λάθους εκτυπώνω απλώς ERROR καθώς είχε ειπωθεί ότι δεν χρειάζεται να εκτυπώνουμε ακριβώς ποιό ήταν το λάθος.

Επέλεξα στο πρόγραμμά μου να απορρίπτονται αρχεία εισόδου τα οποία χρησιμοποιούν δεσμευμένες λέξεις στην java ως ονόματα μεταβλητών όπως char,for κ.ά.