Μεταγλωττιστές

Java JVM

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Java JVM

Η εικονική μηχανή Java είναι μία αφηρημένη μηχανή υπολογισμών.

Όπως μία πραγματική μηχανή, έχει ένα σύνολο εντολών μέσω των οποίων μπορεί να πραγματοποιήσει πράξεις καθώς και να διαχειριστεί διάφορα κομμάτια μνήμης.

Εικονική Μηχανή Java

Η εικονική μηχανή Java δεν έχει καμία γνώση σχετικά με την γλώσσα προγραμματισμού Java.

Γνωρίζει μόνο το πρότυπο αρχείου κλάσης **(class file format)** το οποίο περιέχει εντολές της εικονικής μηχανής (bytecodes), έναν πίνακα συμβόλων καθώς και διάφορες βοηθητικές πληροφορίες.

Προδιαγραφές Εικονικής Μηχανής Java

Οι σχεδιαστές της εικονικής μηχανής Java περιγράφουν τις διάφορες λειτουργίες στο παρακάτω πρότυπο:

- http://docs.oracle.com/javase/specs/jvms/se7/html/index.html

Για να υλοποιησετε μία σωστή εικονική μηχανή Java, αρκεί να διαβάσετε σωστά το πρότυπο αρχείου κλάσης και να μπορεί η μηχανή σας να εκτελεί τις λειτουργίες που περιγράφονται στο παραπάνω κείμενο.

Το πρότυπο της εικονικής μηχανής δεν αναφέρει πουθενά θέματα υλοποίησης, όπως π.χ πως υλοποιείται η κάθε εντολή του JVM με χαμηλότερου επιπέδου εντολές ή πως λειτουργεί ο συλλέκτης σκουπιδιών, κ.τ.λ.

Υλοποιήσεις JVM

Υπάρχουν διάφορες υλοποιήσεις JVM:

1. http://en.wikipedia.org/wiki/List_of_Java_virtual_machines
2. http://en.wikipedia.org/wiki/Free_Java_implementations
3. http://openjdk.java.net/

Τύποι

Όπως και η γλώσσα προγραμματισμού Java, το JVM υποστηρίζει δύο ειδών τύπους:

- βασικούς τύπους (primitive types)
- αναφορές (references)

Το JVM υποθέτει πως το μεγαλύτερο μέρος του έλεγχου τύπων έχει πραγματοποιηθεί κατά την διάρκεια της μεταγλώττισης.

Για αυτό το λόγο οι εντολές της εικονικής μηχανής έχουν εκδόσεις ανά τύπο, π.χ `iadd` για την πρόσθεση ακεραίων και `fadd` για την πρόσθεση αριθμών κινητής υποδιαστολής.

Βασικοί Τύποι

- byte, 8-bit προσημασμένοι ακέραιοι σε συμπλήρωμα ως προς 2
- short, 16-bit προσημασμένοι ακέραιοι σε συμπλήρωμα ως προς 2
- int, 32-bit προσημασμένοι ακέραιοι σε συμπλήρωμα ως προς 2
- long, 64-bit προσημασμένοι ακέραιοι σε συμπλήρωμα ως προς 2
- char, 16-bit μη-προσημασμένοι ακέραιοι σε Unicode
- float, 32-bit single precision IEEE 754
- double, 64-bit double precision IEEE 754
- boolean, true ή false (υλοποιούνται με int και 0 για false, 1 για true)

Τύποι Αναφοράς

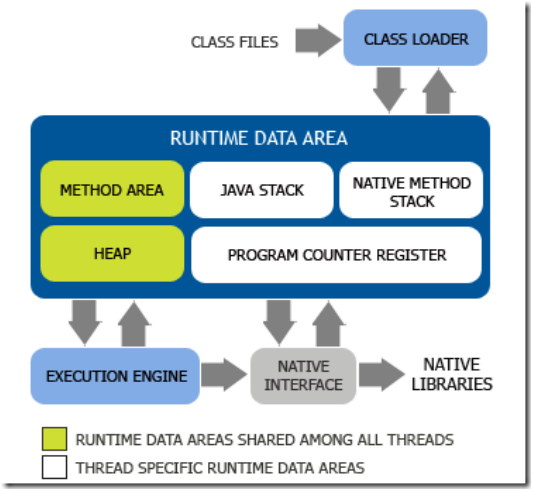
Το JVM παρέχει υποστήριξη για αντικείμενα.

Ένα αντικείμενο είναι είτε ένα δυναμικά δημιουργημένο στιγμιότυπο μίας κλάσης είτε ένας πίνακας.

Η πρόσβαση στα αντικείμενα γίνεται μέσω του τύπου *αναφορά* (reference).

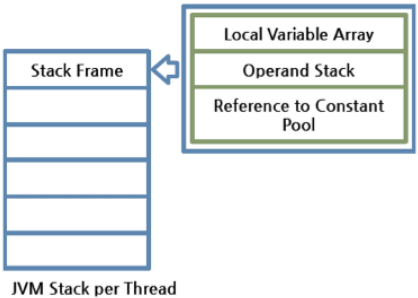
Περιοχές Μνήμης

Η εικονική μηχανή Java περιέχει διάφορες περιοχές δεδομένων που χρησιμοποιούνται κατά την εκτέλεση ενός προγράμματος.



- Μερικές από αυτές δημιουργούνται μόλις ξεκινήσει το JVM και καταστρέφονται μόλις τερματιστεί η εκτέλεση του JVM.
- Άλλες είναι ανά νήμα εκτέλεσης (thread) και άρα δημιουργούνται όταν δημιουργηθεί ένα καινούριο νήμα και καταστρέφονται όταν τερματίσει το νήμα.

Στοίβες του JVM



Κάθε νήμα (thread) ενός JVM έχει μία προσωπική στοίβα, η οποία δημιουργείται την ίδια στιγμή που δημιουργείται και το νήμα.

- Η στοίβα αυτή αποθηκεύει **εγγραφές ενεργοποίησης (frames)**.
- Είναι ανάλογη με την στοίβα κλήσεων άλλων γλωσσών προγραμματισμού όπως η C.
- Αποθηκεύει τοπικές μεταβλητές, μερικά αποτελέσματα και παίζει ρόλο στην κλήση συναρτήσεων και την επιστροφή τιμών από τις συναρτήσεις.

Heap

Το JVM έχει και ένα κομμάτι μνήμης (heap) που διαμοιράζεται μεταξύ όλων των νημάτων του JVM.

Η μνήμη αυτή είναι το μέρος που αποθηκεύονται όλα τα στιγμιότυπα των κλάσεων και των πινάκων που δημιουργούνται κατά την διάρκεια της εκτέλεσης ενός προγράμματος.

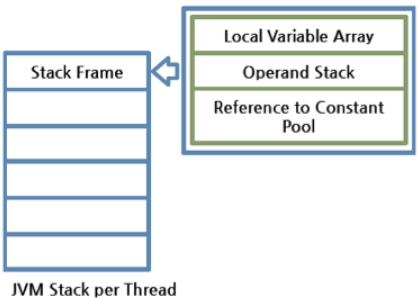
- Ο σωρός δημιουργείται κατά την εκκίνηση της εικονικής μηχανής.
- Ο **συλλέκτης σκουπιδιών (garbage collector)** είναι υπεύθυνος για την συλλογή χώρου στο σωρό που ελευθερώθηκε.
- Η ακριβής συμπεριφορά του συλλέκτη σκουπιδιών είναι στο χέρι της υλοποίησης.

Εγγραφές Ενεργοποίησης

Frames

Η εγγραφή ενεργοποίησης χρησιμοποιείται για να αποθηκευθούν δεδομένα, μερικά αποτελέσματα καθώς και να υλοποιηθεί η κλήση συναρτήσεων.

Μία καινούρια εγγραφή ενεργοποίηση δημιουργείται κάθε φορά που γίνεται μία κλήση συνάρτησης. Η εγγραφή αυτή καταστρέφεται μόλις επιστρέψει η συνάρτηση, είτε κανονικά είτε λόγω κάποιου λάθους.

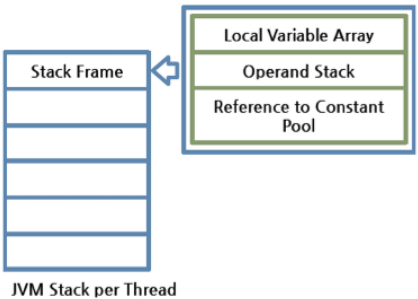


Κάθε εγγραφή ενεργοποίησης περιέχει

1. έναν πίνακα με τοπικές μεταβλητές
2. μία στοίβα ορισμάτων
3. μία αναφορά στις σταθερές της κλάσης που ανήκει η μέθοδος (runtime constant pool)

Εγγραφές Ενεργοποίησης

Frames



Το μέγεθος του πίνακα τοπικών μεταβλητών και της στοίβας ορισμάτων καθορίζονται την ώρα της μεταγλώττισης και παρέχονται μαζί με τον κώδικα της συνάρτησης.

- Μόνο μία εγγραφή είναι ενεργή ανά πάσα στιγμή.
- Οι εντολές που αναφέρονται σε τοπικές μεταβλητές και την στοίβα ορισμάτων, είναι σε σχέση με την ενεργή εγγραφή ενεργοποίησης.
- Μία εγγραφή σταματά να είναι ενεργή εαν η συνάρτηση της καλέσει άλλη συνάρτηση ή αν η συνάρτηση επιστρέψει.

Τοπικές Μεταβλητές

Κάθε εγγραφή ενεργοποίησης περιέχει έναν πίνακα με τοπικές μεταβλητές. Το μέγεθος του πίνακα υπολογίζεται κατά την μεταγλώττιση και παρέχεται μαζί με τον κώδικα της κλάσης.

Οι τοπικές μεταβλητές διευθυνσιοδοτούνται με ακέραιους ξεκινώντας από το μηδέν (0).

- Μία τοπική μεταβλητή μπορεί να αποθηκεύσει μία τιμή τύπου boolean, byte, char, short, int, float, reference.
- Μία τιμή τύπου long ή double χρειάζεται δύο συνεχόμενες τοπικές μεταβλητές.

Το JVM χρησιμοποιεί τοπικές μεταβλητές για να περάσει παραμέτρους κατά την κλήση μεθόδων.

Operand Stack

Κάθε εγγραφή ενεργοποίησης περιέχει μία LIFO στοίβα που ονομάζεται **στοίβα τελεστών**.

Το μέγιστο βάθος της στοίβας αυτής καθορίζεται κατά την μεταγλώττιση και είναι αποθηκευμένο μέσα στον κώδικα.

- Η στοίβα τελεστών είναι άδεια μόλις δημιουργηθεί η εγγραφή.

Operand Stack

Κάθε εγγραφή ενεργοποίησης περιέχει μία LIFO στοίβα που ονομάζεται **στοίβα τελεστών**.

Το μέγιστο βάθος της στοίβας αυτής καθορίζεται κατά την μεταγλώττιση και είναι αποθηκευμένο μέσα στον κώδικα.

- Η στοίβα τελεστών είναι άδεια μόλις δημιουργηθεί η εγγραφή.
- Το JVM έχει οδηγίες (instructions) για να φορτωθούν σταθερές τιμές, τιμές τοπικών μεταβλητών ή τιμές μεταβλητών κλάσης στην στοίβα τελεστών.

Operand Stack

Κάθε εγγραφή ενεργοποίησης περιέχει μία LIFO στοίβα που ονομάζεται **στοίβα τελεστών**.

Το μέγιστο βάθος της στοίβας αυτής καθορίζεται κατά την μεταγλώττιση και είναι αποθηκευμένο μέσα στον κώδικα.

- Η στοίβα τελεστών είναι άδεια μόλις δημιουργηθεί η εγγραφή.
- Το JVM έχει οδηγίες (instructions) για να φορτωθούν σταθερές τιμές, τιμές τοπικών μεταβλητών ή τιμές μεταβλητών κλάσης στην στοίβα τελεστών.
- Άλλες οδηγίες διαβάζουν ορίσματα από την στοίβα, κάνουν κάποιες πράξεις και αποθηκεύουν το αποτέλεσμα στην στοίβα.

Operand Stack

Κάθε εγγραφή ενεργοποίησης περιέχει μία LIFO στοίβα που ονομάζεται **στοίβα τελεστών**.

Το μέγιστο βάθος της στοίβας αυτής καθορίζεται κατά την μεταγλώττιση και είναι αποθηκευμένο μέσα στον κώδικα.

- Η στοίβα τελεστών είναι άδεια μόλις δημιουργηθεί η εγγραφή.
- Το JVM έχει οδηγίες (instructions) για να φορτωθούν σταθερές τιμές, τιμές τοπικών μεταβλητών ή τιμές μεταβλητών κλάσης στην στοίβα τελεστών.
- Άλλες οδηγίες διαβάζουν ορίσματα από την στοίβα, κάνουν κάποιες πράξεις και αποθηκεύουν το αποτέλεσμα στην στοίβα.
- Η στοίβα χρησιμοποιείται επίσης για να προετοιμαστούν οι παράμετροι κατά την κλήση μιας μεθόδου καθώς και για την επιστροφή των αποτελεσμάτων.

Operand Stack

Κάθε εγγραφή ενεργοποίησης περιέχει μία LIFO στοίβα που ονομάζεται **στοίβα τελεστών**.

Το μέγιστο βάθος της στοίβας αυτής καθορίζεται κατά την μεταγλώττιση και είναι αποθηκευμένο μέσα στον κώδικα.

- Η στοίβα τελεστών είναι άδεια μόλις δημιουργηθεί η εγγραφή.
- Το JVM έχει οδηγίες (instructions) για να φορτωθούν σταθερές τιμές, τιμές τοπικών μεταβλητών ή τιμές μεταβλητών κλάσης στην στοίβα τελεστών.
- Άλλες οδηγίες διαβάζουν ορίσματα από την στοίβα, κάνουν κάποιες πράξεις και αποθηκεύουν το αποτέλεσμα στην στοίβα.
- Η στοίβα χρησιμοποιείται επίσης για να προετοιμαστούν οι παράμετροι κατά την κλήση μιας μεθόδου καθώς και για την επιστροφή των αποτελεσμάτων.
- Μερική τύποι δεδομένων π.χ long, double καταναλώνουν δύο θέσεις στην κορυφή της στοίβας.

Operand Stack

Η οδηγία `iadd` λέει στο JVM να αφαιρέσει δύο ακέραιους από την κορυφή της στοίβας, να τους προσθέσει και στην συνέχεια να εισάγει το αποτέλεσμα στην στοίβα.

Αντίστοιχες οδηγίες υπάρχουν για άλλους τύπους, π.χ `fadd`, `dadd`, κ.α.

Οι τιμές στην στοίβα πρέπει να χρησιμοποιούνται από εντολές κατάλληλες με τον τύπο τους. Δεν είναι π.χ δυνατό να κάνουμε push δύο ακέραιους και να τους χρησιμοποιήσουμε ως έναν long.

Οδηγίες JVM

Οι οδηγίες τους JVM αποτελούνται από ένα **byte opcode** που καθορίζει την λειτουργία που θα πραγματοποιηθεί, ακολουθούμενο από μηδέν ή περισσότερες παραμέτρους.

- Πολλές λειτουργίες του JVM αποτελούνται από μόνο ένα byte.
- Αυτή η επιλογή έχει γίνει με σκοπό να είναι ο κώδικας μικρός σε μέγεθος.

Οδηγίες JVM

Οι περισσότερες οδηγίες του JVM περιέχουν στο opcode και πληροφορία σχετικά με τον τύπο των δεδομένων.

- Για παράδειγμα η λειτουργία `iload` φορτώνει το περιεχόμενο μίας τοπικής μεταβλητής που πρέπει να είναι ακέραιος, στην στοίβα τελεστών.
- Αντίστοιχα η λειτουργία `fload` για τύπο float.

Παρόλο που η υλοποίηση είναι ίδια, υπάρχουν δύο ξεχωριστά opcodes.

Οδηγίες JVM

Οι περισσότερες οδηγίες του JVM περιέχουν στο opcode και πληροφορία σχετικά με τον τύπο των δεδομένων.

Για τις περισσότερες λειτουργίες που αφορούν τύπους, ο τύπος κωδικοποιείται στον κωδικό της εντολής (opcode) με ένα γράμμα.

- i: για λειτουργία με int
- l: για λειτουργία με long
- s: για λειτουργία με short
- b: για λειτουργία με byte
- c: για λειτουργία με char
- f: για λειτουργία με float
- d: για λειτουργία με double
- a: για λειτουργία με reference

Οι εντολές αυτές μεταφέρουν τιμές μεταξύ των τοπικών μεταβλητών και της στοίβας τελεστών που βρίσκονται στην ενεργή εγγραφή ενεργοποίησης (frame).

`iload i`	φορτώνει την τοπική μεταβλητή *i* στην στοίβα
`iload_0`	φορτώνει την τοπική μεταβλητή 0 στην στοίβα
`iload_1`	φορτώνει την τοπική μεταβλητή 1 στην στοίβα
`iload_2`	φορτώνει την τοπική μεταβλητή 2 στην στοίβα
`iload_3`	φορτώνει την τοπική μεταβλητή 3 στην στοίβα

Οι εκδόσεις χωρίς παράμετρο κωδικοποιούν με μόνο ένα byte τις εκδόσεις της εντολής που χρησιμοποιούνται πολύ συχνά.

Η τοπική μεταβλητή πρέπει να είναι **int**.

Οι εντολές αυτές μεταφέρουν τιμές μεταξύ των τοπικών μεταβλητών και της στοίβας τελεστών που βρίσκονται στην ενεργή εγγραφή ενεργοποίησης (frame).

`lload i`	φορτώνει την τοπική μεταβλητή *i* στην στοίβα
`lload_0`	φορτώνει την τοπική μεταβλητή 0 στην στοίβα
`lload_1`	φορτώνει την τοπική μεταβλητή 1 στην στοίβα
`lload_2`	φορτώνει την τοπική μεταβλητή 2 στην στοίβα
`lload_3`	φορτώνει την τοπική μεταβλητή 3 στην στοίβα

Οι εκδόσεις χωρίς παράμετρο κωδικοποιούν με μόνο ένα byte τις εκδόσεις της εντολής που χρησιμοποιούνται πολύ συχνά.

Η τοπική μεταβλητή πρέπει να είναι **long**.

Οι εντολές αυτές μεταφέρουν τιμές μεταξύ των τοπικών μεταβλητών και της στοίβας τελεστών που βρίσκονται στην ενεργή εγγραφή ενεργοποίησης (frame).

`fload i`	φορτώνει την τοπική μεταβλητή *i* στην στοίβα
`fload_0`	φορτώνει την τοπική μεταβλητή 0 στην στοίβα
`fload_1`	φορτώνει την τοπική μεταβλητή 1 στην στοίβα
`fload_2`	φορτώνει την τοπική μεταβλητή 2 στην στοίβα
`fload_3`	φορτώνει την τοπική μεταβλητή 3 στην στοίβα

Οι εκδόσεις χωρίς παράμετρο κωδικοποιούν με μόνο ένα byte τις εκδόσεις της εντολής που χρησιμοποιούνται πολύ συχνά.

Η τοπική μεταβλητή πρέπει να είναι **float**.

Οι εντολές αυτές μεταφέρουν τιμές μεταξύ των τοπικών μεταβλητών και της στοίβας τελεστών που βρίσκονται στην ενεργή εγγραφή ενεργοποίησης (frame).

`dload i`	φορτώνει την τοπική μεταβλητή *i* στην στοίβα
`dload_0`	φορτώνει την τοπική μεταβλητή 0 στην στοίβα
`dload_1`	φορτώνει την τοπική μεταβλητή 1 στην στοίβα
`dload_2`	φορτώνει την τοπική μεταβλητή 2 στην στοίβα
`dload_3`	φορτώνει την τοπική μεταβλητή 3 στην στοίβα

Οι εκδόσεις χωρίς παράμετρο κωδικοποιούν με μόνο ένα byte τις εκδόσεις της εντολής που χρησιμοποιούνται πολύ συχνά.

Η τοπική μεταβλητή πρέπει να είναι **double**.

Οι εντολές αυτές μεταφέρουν τιμές μεταξύ των τοπικών μεταβλητών και της στοίβας τελεστών που βρίσκονται στην ενεργή εγγραφή ενεργοποίησης (frame).

`aload i`	φορτώνει την τοπική μεταβλητή *i* στην στοίβα
`aload_0`	φορτώνει την τοπική μεταβλητή 0 στην στοίβα
`aload_1`	φορτώνει την τοπική μεταβλητή 1 στην στοίβα
`aload_2`	φορτώνει την τοπική μεταβλητή 2 στην στοίβα
`aload_3`	φορτώνει την τοπική μεταβλητή 3 στην στοίβα

Οι εκδόσεις χωρίς παράμετρο κωδικοποιούν με μόνο ένα byte τις εκδόσεις της εντολής που χρησιμοποιούνται πολύ συχνά.

Η τοπική μεταβλητή πρέπει να είναι **reference**.

Οι εντολές αυτές μεταφέρουν τιμές μεταξύ των τοπικών μεταβλητών και της στοίβας τελεστών που βρίσκονται στην ενεργή εγγραφή ενεργοποίησης (frame).

`istore i`	αποθηκεύει την κορυφή της στοίβας στην τοπική μεταβλητή *i*
`istore_0`	αποθηκεύει την κορυφή της στοίβας στην τοπική μεταβλητή 0
`istore_1`	αποθηκεύει την κορυφή της στοίβας στην τοπική μεταβλητή 1
`istore_2`	αποθηκεύει την κορυφή της στοίβας στην τοπική μεταβλητή 2
`istore_3`	αποθηκεύει την κορυφή της στοίβας στην τοπική μεταβλητή 3

Οι εκδόσεις χωρίς παράμετρο κωδικοποιούν με μόνο ένα byte τις εκδόσεις της εντολής που χρησιμοποιούνται πολύ συχνά.

Η τοπική μεταβλητή πρέπει να είναι **int**.

Αντίστοιχα με την load υπάρχουν οδηγίες `lstore`, `lstore_<n>`, `fstore`, `fstore_<n>`, `dstore`, `dstore_<n>`, `astore`, `astore_<n>`.

Οδηγίες Φόρτωσης και Αποθήκευσης

Load/Store

Υπάρχουν επίσης εντολές φόρτωσης σταθερών (constants) στην στοίβα.

`bipush`	αποθηκεύει ένα byte (παράμετρο) στην στοίβα
`sipush`	αποθηκεύει ένα short (παράμετρο) στην στοίβα
`ldc`	φορτώνει μία σταθερά (int, float, String) στην στοίβα παίρνει ως παράμετρο την διεύθυνση της σταθεράς
`ldc2_w`	φορτώνει μία σταθερά (long, double) στην στοίβα παίρνει ως παράμετρο την διεύθυνση της σταθεράς

και άλλες όπως

`aconst_null`	αποθηκεύει ένα null reference στην στοίβα
`iconst_m1`	αποθηκεύει -1 στην στοίβα
`iconst_i`	αποθηκεύει τον ακέραιο *i* στην στοίβα
`lconst_l`	αποθηκεύει τον long *l* στην στοίβα
`fconst_f`	αποθηκεύει τον float *f* στην στοίβα
`dconst_d`	αποθηκεύει τον double *d* στην στοίβα

Αριθμητικές Οδηγίες

Οι οδηγίες για αριθμητικές πράξεις συνήθως χρησιμοποιούν ως ορίσματα τις δύο τιμές στην κορυφή της στοίβας και αποθηκεύουν το αποτέλεσμα στην κορυφή της στοίβας.

πρόσθεση	`iadd`, `ladd`, `fadd`, `dadd`
αφαίρεση	`isub`, `lsub`, `fsub`, `dsub`
πολλαπλασιασμός	`imul`, `lmul`, `fmul`, `dmul`
διαίρεση	`idiv`, `ldiv`, `fdiv`, `ddiv`
υπόλοιπο	`irem`, `lrem`, `frem`, `drem`
αντίθετος	`ineg`, `lneg`, `fneg`, `dneg`
ολίσθηση	`ishl`, `ishr`, `iushr`, `lshl`, `lshr`, `lushr`
ανά-bit OR	`ior`, `lor`
ανά-bit AND	`iand`, `land`
ανά-bit XOR	`ixor`, `lxor`
αύξηση τοπικής μεταβλητής	`iinc`
σύγκριση	`dcmpg`, `dcmpl`, `fcmpg`, `fcmpl`, `lcmp`

Αλλαγή Τύπων

`i2l`	μετατρέπει την κορυφή της στοίβας από int σε long
`i2f`	μετατρέπει την κορυφή της στοίβας από int σε float
`i2d`	μετατρέπει την κορυφή της στοίβας από int σε double
`l2f`	μετατρέπει την κορυφή της στοίβας από long σε float
`l2d`	μετατρέπει την κορυφή της στοίβας από long σε double
`f2d`	μετατρέπει την κορυφή της στοίβας από float σε double

και

`i2b`	μετατρέπει την κορυφή της στοίβας από int σε byte
`i2c`	μετατρέπει την κορυφή της στοίβας από int σε char
`i2s`	μετατρέπει την κορυφή της στοίβας από int σε short
`l2i`	μετατρέπει την κορυφή της στοίβας από long σε int
`f2i`	μετατρέπει την κορυφή της στοίβας από float σε int
`f2l`	μετατρέπει την κορυφή της στοίβας από float σε long
`d2i`	μετατρέπει την κορυφή της στοίβας από double σε int
`d2l`	μετατρέπει την κορυφή της στοίβας από double σε long
`d2f`	μετατρέπει την κορυφή της στοίβας από double σε float

Δημιουργία Αντικειμένων και Πινάκων

Ενώ και τα στιγμιότυπα κλάσεων αλλά και οι πίνακες είναι αντικείμενα, υπάρχουν ξεχωριστές οδηγίες διαχείρισης τους.

δημιουργία αντικείμενων	`new`
δημιουργία πινάκων	`newarray`, `anewarray` `multianewarray`
πρόσβαση σε μεταβλητές αντικειμένου	`getfield`, `putfield`
πρόσβαση σε μεταβλητές κλάσης	`getstatic`, `putstatic`

φόρτωση στοιχείου πίνακα στην στοίβα	`baload`, `caload`, `saload`
	`iaload`, `laload`, `faload`
	`daload`, `aaload`

αποθήκευση σε στοιχείο πίνακα	`bastore`, `castore`, `sastore`
	`iastore`, `lastore`, `fastore`
	`dastore`, `aastore`

μήκος πίνακα	`arraylength`
έλεγχος ιδιοτήτων αντικειμένων	`instanceof`, `checkcast`

Οδηγίες Στοίβας

Ορισμένες λειτουργίες προσφέρονται για την απευθείας διαχείριση της στοίβας.

`pop`	αφαιρεί την τιμή από την κορυφή της στοίβας
`dup`	προσθέτει την τιμή της κορυφής της στοίβας άλλη μία φορά στην στοίβα
`swap`	αντιστρέφει τις δύο λέξεις στην κορυφή της στοίβας (οι λέξεις δεν πρέπει να είναι long ή double)

καθώς και άλλες

`pop2`, `dup2`, `dup_x1`, `dup2_x1`, `dup_x2`, `dup2_x2`

Αλλαγή Ελέγχου

Υπάρχουν διάφορες εκδόσεις εντολών για την αλλαγή ελέγχου στο JVM.

- Οι παρακάτω ελέγχουν μία ακέραια τιμή στην κορυφή της στοίβας σε σχέση με το μηδέν (0) και αλλάζουν τον έλεγχο.
  `ifeq`, `iflt`, `ifle`, `ifne`, `ifgt`, `ifge`,
- Έλεγχος εαν η κορυφή της στοίβας περιέχει null ή όχι.
  `ifnull`, `ifnonnull`
- Σύγκριση δύο ακεραίων στην κορυφή της στοίβας.
  `if_icmpeq`, `if_icmpne`, `if_icmplt`, `if_icmpgt`, `if_icmple`, `if_icmpge`
- Σύγκριση δύο references στην κορυφή της στοίβας.
  `if_acmpeq`, `if_acmpne`
- `tableswitch`, `lookupswitch`
- `goto`, `goto_w`, `jsr`, `jsr_w`, `ret`.

Για την αλλαγή ελέγχου μέσω σύγκρισης τιμών τύπου long, float ή double, πρώτα πρέπει να χρησιμοποιηθεί μία εντολή που συγκρίνει τα δύο ορίσματα και τοποθετεί στην στοίβα έναν ακέραιο με την τιμή της σύγκρισης.

Οδηγίες Κλήσης Μεθόδων

κλήση μεθόδου αντικειμένου	`invokevirtual`
κλήση μεθόδου interface	`invokeinterface`
κλήση constructor, private μεθόδων και μέθοδο υπερ-κλάσης	`invokespecial`
κλήση στατικών μεθόδων	`invokestatic`

και για επιστροφή τιμών

`return`, `ireturn`, `lreturn`, `freturn`, `dreturn`, `areturn`

Οδηγίες Κλήσης Μεθόδων

κλήση μεθόδου αντικειμένου	`invokevirtual`
κλήση μεθόδου interface	`invokeinterface`
κλήση constructor, private μεθόδων και μέθοδο υπερ-κλάσης	`invokespecial`
κλήση στατικών μεθόδων	`invokestatic`

και για επιστροφή τιμών

`return`, `ireturn`, `lreturn`, `freturn`, `dreturn`, `areturn`

Στην έκδοση 1.7 του JVM προστέθηκε και μία οδηγία `invokedynamic`, η οποία επιτρέπει την εύρεση της μεθόδου κατά την εκτέλεση του προγράμματος.

Κλήση Μεθόδων και Επιστροφή Αποτελέσματος

Κλήση σε μέθοδο ενός αντικειμένου χρειάζεται

- εισαγωγή της αναφοράς του αντικειμένου στην στοίβα
- εισαγωγή όλων των παραμέτρων στην στοίβα
- χρήση της αντίστοιχης οδηγίας, π.χ `invokevirtual` ή `invokeinterface`

Σε περίπτωση που η μέθοδος είναι static δεν προσθέτουμε κάποια αναφορά στην στοίβα αφού δεν υπάρχει instance αντικειμένου αλλά μόνο τις παραμέτρους και χρησιμοποιούμε την οδηγία `invokestatic`.

Με το πέρας της εκτέλεσης μίας μεθόδου, το αποτέλεσμα της μεθόδου βρίσκεται στην κορυφή της στοίβας.

Παράμετροι Μεθόδων και αναφορά this

Όταν ξεκινάει η εκτέλεση μίας μεθόδου οι παράμετροι βρίσκονται

- η αναφορά `this` βρίσκεται στην τοπική μεταβλητή 0,
- οι παράμετροι βρίσκονται με την σειρά εμφάνισης τους στις επόμενες τοπικές μεταβλητές
- προσοχή χρειάζεται στην περίπτωση των τύπων long και double που χρησιμοποιούν δύο συνεχόμενες θέσεις στον πίνακα τοπικών μεταβλητών

Σε περίπτωση που η μέθοδος είναι static, η πρώτη παράμετρος βρίσκεται στην τοπική μεταβλητή 0.

Άλλες Οδηγίες

Υπάρχουν και άλλες οδηγίες όπως για παράδειγμα υποστήριξη για λάθη (exceptions) καθώς και υποστήριξη για συγχρονισμό (monitors).

Για περισσότερες πληροφορίες μπορείτε να δείτε τα

- https://docs.oracle.com/javase/specs/jvms/se8/html/index.html
- https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-6.html
- http://en.wikipedia.org/wiki/Java_bytecode_instruction_listings
- https://docs.oracle.com/javase/specs/

Για ιστορικούς λόγους τα ονόματα κλάσεων και interfaces μέσα στα αρχεία κλάσεων εμφανίζονται λίγο διαφορετικά.

Αντί για την χρήση τελείας ”.” όπως π.χ

```
java.lang.Thread
```

χρησιμοποιείται ASCII forward slash ”/” και άρα προκύπτει

```
java/lang/Thread
```

Οι τύποι των μελών μίας κλάσης αναπαρίστανται από **περιγραφές (descriptors)** οι οποίες ακολουθούν πολύ συγκεκριμένους κανόνες.

Ο παρακάτω πίνακας περιγράφει αυτούς τους κανόνες:

BaseType Character	Type	Interpretation
B	byte	signed byte
C	char	Unicode character
D	double	double-precision floating-point value
F	float	single-precision floating-point value
I	int	integer
J	long	long integer
L<classname>;	reference	an instance of class <classname>
S	short	signed short
Z	boolean	true or false
[	reference	one array dimension

Με την παράθεση αυτών των κανόνων μπορούμε να αναπαραστήσουμε πιο σύνθετους τύπους όπως π.χ

```
long x                    J
double d[][][]           [[[D
String s        Ljava/lang/String;
String s[][]    [[Ljava/lang/String;
```

Οι τύποι των μεθόδων μίας κλάσης αναπαρίστανται με τους τύπους των παραμέτρων σε παρένθεση και στην συνέχεια είτε με την χρήση του χαρακτήρα `V` για μία συνάρτηση που επιστρέφει `void` είτε με τον τύπο επιστροφής.

π.χ

- `Object mymethod(int i, double d, Thread t)`

  με

  `(IDLjava/lang/Thread;)Ljava/lang/Object;`

- `void main(String args[])`

  με

  `([Ljava/lang/String;)V`

javap: Java Class File Disassembler

Το εργαλείο **javap** μας επιτρέπει να κοιτάξουμε ένα αρχείο κλάσης στο επίπεδο bytecode.

Φτιάξτε ένα αρχείο `HelloWorld.java` με το παρακάτω απλό πρόγραμμα:

```
import java.lang.System;

public class HelloWorld {

    public static void main(String args[]) {
        System.out.println("Hello world!");
    }

}
```

Μπορούμε να μεταγλωττίσουμε εκτελώντας:

```
> javac HelloWorld.java
```

Το αποτέλεσμα είναι ένα αρχείο `HelloWorld.class` το οποίο μπορούμε να εκτελέσουμε με:

```
> java HelloWorld
```

javap: Java Class File Disassembler

Εκτελώντας

```
> javap -c HelloWorld.class
```

πέρνουμε την εξής έξοδο:

```
Compiled from "HelloWorld.java"
public class HelloWorld {
 public HelloWorld();
 Code:
  0: aload_0
  1: invokespecial #1   // Method java/lang/Object."<init>":()V
  4: return

 public static void main(java.lang.String[]);
 Code:
  0: getstatic     #2   // Field java/lang/System.out:Ljava/io/PrintStream;
  3: ldc           #3   // String Hello world!
  5: invokevirtual #4   // Method java/io/PrintStream.println:(Ljava/lang/String;)V
  8: return
}
```

Οι εντολές περιέχουν αναφορές στις σταθερές που είναι αποθηκευμένες στο αρχείο κλάσης, και δίπλα ακριβώς σε σχόλιο περιγράφεται κάθε τέτοια εγγραφή.

javap: Java Class File Disassembler

Για περισσότερες λεπτομέρειες όπως π.χ private μεθόδους, καθώς και τον πίνακα με τις σταθερές

```
> javap -c -p -verbose HelloWorld.class
```

javap: Java Class File Disassembler

Το πρόγραμμα

```
import java.lang.System;

public class Loop {

    public static void main(String args[]) {
        new Loop().printRange(100);
    }

    public void printRange(int i) {
        while( i > 0 ) {
            System.out.println(i);
            i--;
        }
    }

}
```

javap: Java Class File Disassembler

γίνεται

```
Compiled from "Loop.java"
public class Loop {
  public Loop();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: new           #2                  // class Loop
       3: dup
       4: invokespecial #3                  // Method "<init>":()V
       7: bipush        100
       9: invokevirtual #4                  // Method printRange:(I)V
      12: return

  public void printRange(int);
    Code:
       0: iload_1
       1: ifle          17
       4: getstatic     #5                  // Field java/lang/System.out:Ljava/io/PrintStream;
       7: iload_1
       8: invokevirtual #6                  // Method java/io/PrintStream.println:(I)V
      11: iinc          1, -1
      14: goto          0
      17: return
}
```

Βιβλιοθήκη ASM

http://asm.ow2.org

Η βιβλιοθήκη ASM παρέχει μεγάλη βοήθεια στην παραγωγή bytecode. Με τον κώδικα που ακολουθεί παράγουμε την κλάση Loop.

```
1  import java.io.FileOutputStream;
2  import java.io.IOException;
3  import java.io.PrintWriter;
4  import java.util.Vector;
5
6  import org.objectweb.asm.ClassWriter;
7  import org.objectweb.asm.Opcodes;
8  import org.objectweb.asm.tree.AbstractInsnNode;
9  import org.objectweb.asm.tree.ClassNode;
10 import org.objectweb.asm.tree.FieldInsnNode;
11 import org.objectweb.asm.tree.IincInsnNode;
12 import org.objectweb.asm.tree.InsnNode;
13 import org.objectweb.asm.tree.IntInsnNode;
14 import org.objectweb.asm.tree.JumpInsnNode;
15 import org.objectweb.asm.tree.LabelNode;
16 import org.objectweb.asm.tree.MethodInsnNode;
17 import org.objectweb.asm.tree.MethodNode;
18 import org.objectweb.asm.tree.TypeInsnNode;
19 import org.objectweb.asm.tree.VarInsnNode;
20 import org.objectweb.asm.util.TraceClassVisitor;
21
22 public class App {
23
24   public static void main(String args[]) {
25     try {
26       new App().generate();
27     }
28     catch(IOException e) {
29       e.printStackTrace();
30     }
31   }
```

Βιβλιοθήκη ASM

http://asm.ow2.org

Στο παράδειγμα αυτό χρησιμοποιούμε το δενδρικό μοντέλο της βιβλιοθήκης. Η βιβλιοθήκη παρέχει και ένα πιο αποδοτικό τρόπο (όσο αναφορά την μνήμη και την ταχύτητα) χρησιμοποιώντας ένα event-driven μοντέλο.

```
33    public void generate() throws IOException {
34      ClassNode cn = new ClassNode();
35      cn.access = Opcodes.ACC_PUBLIC;
36      cn.version = Opcodes.V1_5;
37      cn.name = "Loop";
38      cn.sourceFile = "Loop.java";
39      cn.superName = "java/lang/Object";
40
41      MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC, "<init>", "()V", null, null);
42      mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
43      mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "java/lang/Object", "<init>", "()V"));
44      mn.instructions.add(new InsnNode(Opcodes.RETURN));
45      mn.maxLocals = 1;
46      mn.maxStack = 1;
47      cn.methods.add(mn);
48
49      mn = new MethodNode(Opcodes.ACC_PUBLIC + Opcodes.ACC_STATIC, "main", "([Ljava/lang/String;)V", null, null);
50      mn.instructions.add(new TypeInsnNode(Opcodes.NEW, "Loop"));
51      mn.instructions.add(new InsnNode(Opcodes.DUP));
52      mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "Loop", "<init>", "()V"));
53      mn.instructions.add(new IntInsnNode(Opcodes.BIPUSH, 100));
54      mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "Loop", "loopRange", "(I)V"));
55      mn.instructions.add(new InsnNode(Opcodes.RETURN));
56      mn.maxLocals = 1;
57      mn.maxStack = 3;
58      cn.methods.add(mn);
```

Βιβλιοθήκη ASM

http://asm.ow2.org

```
60
61          mn = new MethodNode(Opcodes.ACC_PUBLIC, "loopRange", "(I)V", null, null);
62          LabelNode l0 = new LabelNode();
63          mn.instructions.add(l0);
64          mn.instructions.add(new VarInsnNode(Opcodes.ILOAD, 1));
65          LabelNode l1 = new LabelNode();
66          mn.instructions.add(new JumpInsnNode(Opcodes.IFLE, l1));
67          mn.instructions.add(new FieldInsnNode(Opcodes.GETSTATIC, "java/lang/System", "out", "Ljava/io/PrintStream;"));
68          mn.instructions.add(new VarInsnNode(Opcodes.ILOAD, 1));
69          mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/io/PrintStream", "println", "(I)V"));
70          mn.instructions.add(new IincInsnNode(1, -1));
71          mn.instructions.add(new JumpInsnNode(Opcodes.GOTO, l0));
72          mn.instructions.add(l1);
73          mn.instructions.add(new InsnNode(Opcodes.RETURN));
74          mn.maxLocals = 2;
75          mn.maxStack = 2;
76          cn.methods.add(mn);
77
78          ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_MAXS);
79          TraceClassVisitor cv = new TraceClassVisitor(cw, new PrintWriter(System.out));
80          cn.accept(cv);
81          byte code[] = cw.toByteArray();
82          FileOutputStream fos = new FileOutputStream("Loop.class");
83          fos.write(code);
84          fos.close();
85      }
86
87  }
```

Η βιβλιοθήκη παρέχει επίσης και ένα εργαλείο **ASMifier** το οποίο δέχεται ως είσοδο ένα αρχείο κλάσης (.class) και παράγει τον απαραίτητο κώδικα για την παραγωγή αυτής της κλάσης.

Εκτελώντας το εργαλείο αυτό με

```
> java -classpath asm-4.2.jar:asm-util-4.2.jar:. org.objectweb.asm.util.ASMifier Loop.class
```

πέρνουμε ως έξοδο τον αντίστοιχο κώδικα παραγωγής της κλάσης με την χρήση του event-driven μοντέλου παραγωγής.

Η μετάφραση από το event-driver μοντέλο στο δενδρικό μοντέλο είναι σχεδόν 1-1. Το δενδρικό μοντέλο καταναλώνει περισσότερο χώρο, αλλά βολεύει σε πολλές περιπτώσεις γιατί δεν επιβάλλει συγκεκριμένη σειρά παραγωγής.

Βιβλιοθήκη ASM

http://asm.ow2.org

```
1   import java.util.*;
2   import org.objectweb.asm.*;
3   import org.objectweb.asm.attrs.*;
4
5   public class LoopDump implements Opcodes {
6
7     public static byte[] dump () throws Exception {
8
9       ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_MAXS);
10      FieldVisitor fv;
11      MethodVisitor mv;
12      AnnotationVisitor av0;
13
14      cw.visit(V1_7, ACC_PUBLIC + ACC_SUPER, "Loop", null, "java/lang/Object", null);
15
16      {
17        mv = cw.visitMethod(ACC_PUBLIC, "<init>", "()V", null, null);
18        mv.visitCode();
19        mv.visitVarInsn(ALOAD, 0);
20        mv.visitMethodInsn(INVOKESPECIAL, "java/lang/Object", "<init>", "()V");
21        mv.visitInsn(RETURN);
22        mv.visitMaxs(1, 1);
23        mv.visitEnd();
24      }
25
26      {
27        mv = cw.visitMethod(ACC_PUBLIC + ACC_STATIC, "main", "([Ljava/lang/String;)V", null, null);
28        mv.visitCode();
29        mv.visitTypeInsn(NEW, "Loop");
30        mv.visitInsn(DUP);
31        mv.visitMethodInsn(INVOKESPECIAL, "Loop", "<init>", "()V");
32        mv.visitIntInsn(BIPUSH, 100);
33        mv.visitMethodInsn(INVOKEVIRTUAL, "Loop", "printRange", "(I)V");
34        mv.visitInsn(RETURN);
35        mv.visitMaxs(2, 1);
36        mv.visitEnd();
37
```

Βιβλιοθήκη ASM

http://asm.ow2.org

```
38          {
39            mv = cw.visitMethod(ACC_PUBLIC, "printRange", "(I)V", null, null);
40            mv.visitCode();
41            Label l0 = new Label();
42            mv.visitLabel(l0);
43            mv.visitFrame(Opcodes.F_SAME, 0, null, 0, null);
44            mv.visitVarInsn(ILOAD, 1);
45            Label l1 = new Label();
46            mv.visitJumpInsn(IFLE, l1);
47            mv.visitFieldInsn(GETSTATIC, "java/lang/System", "out", "Ljava/io/PrintStream;");
48            mv.visitVarInsn(ILOAD, 1);
49            mv.visitMethodInsn(INVOKEVIRTUAL, "java/io/PrintStream", "println", "(I)V");
50            mv.visitIincInsn(1, -1);
51            mv.visitJumpInsn(GOTO, l0);
52            mv.visitLabel(l1);
53            mv.visitFrame(Opcodes.F_SAME, 0, null, 0, null);
54            mv.visitInsn(RETURN);
55            mv.visitMaxs(2, 2);
56            mv.visitEnd();
57          }
58
59          cw.visitEnd();
60          return cw.toByteArray();
61        }
62      }
```

ASM tree API

- ClassNode
- MethodInsnNode
- FieldInsnNode
- InsnNode
- LdcInsnNode
- LabelInsnNode
- JumpInsnNode
- TypeInsnNode
- VarInsnNode
- IincInsnNode

ASM tree API

- LineNumberNode
- LookupSwitchInsnNode
- MultiANewArrayInsnNode
- TableSwitchInsnNode
- InvokeDynamicInsnNode
- FrameNode

Οδηγίες και περιγραφή της βιβλιοθήκης ASM μπορείτε να βρείτε εδώ:
http://download.forge.objectweb.org/asm/asm4-guide.pdf

Παράδειγμα Κατασκευής με την ASM

Κλάση Complex

- Θα χρησιμοποιήσουμε ως παράδειγμα μία υλοποίηση μιγαδικών αριθμών.
- Η υλοποίηση της κλάσης που ακολουθεί είναι μία μικρότερη και ελαφρώς παραλλαγμένη έκδοση από τον παρακάτω σύνδεσμο.
- http://introcs.cs.princeton.edu/java/97data/Complex.java.html.

```
1   public class Complex {
2
3       private double re;   // the real part
4       private double im;   // the imaginary part
5
6       public Complex(double re, double im) {
7           this.re = re;
8           this.im = im;
9       }
10
11      public String toString() {
12          if (im == 0) {
13              return re + "";
14          }
15          if (re == 0) {
16              return im + "i";
17          }
18          if (im < 0) {
19              return re + " - " + (-im) + "i";
20          }
21          return re + " + " + im + "i";
22      }
```

Παράδειγμα Κατασκευής με την ASM

Κλάση Complex

```
23
24          // return a new Complex object whose value is (this + b)
25          public Complex plus(Complex b) {
26              double real = this.re + b.re;
27              double imag = this.im + b.im;
28              return new Complex(real, imag);
29          }
30
31          // return a new Complex object whose value is (this - b)
32          public Complex minus(Complex b) {
33              double real = this.re - b.re;
34              double imag = this.im - b.im;
35              return new Complex(real, imag);
36          }
37
38          // return a new Complex object whose value is (this * b)
39          public Complex times(Complex b) {
40              double real = this.re * b.re - this.im * b.im;
41              double imag = this.re * b.im + this.im * b.re;
42              return new Complex(real, imag);
43          }
44
45          // scalar multiplication
46          // return a new object whose value is (this * alpha)
47          public Complex times(double alpha) {
48              return new Complex(alpha * re, alpha * im);
49          }
50
51          public double abs() {
52              // Math.sqrt(re*re + im*im)
53              return Math.hypot(re, im);
54          }
55
56          // return the real or imaginary part
57          public double re() {
58              return re;
59
```

Παράδειγμα Κατασκευής με την ASM

Κλάση Complex

```
60
61      public double im() {
62          return im;
63      }
64
65      // return a new Complex object whose value is the reciprocal of this
66      public Complex reciprocal() {
67          double scale = re * re + im * im;
68          return new Complex(re / scale, -im / scale);
69      }
70
71      // return a / b
72      public Complex divides(Complex b) {
73          return this.times(b.reciprocal());
74      }
75
76      // a static version of plus
77      public static Complex plus(Complex a, Complex b) {
78          double real = a.re + b.re;
79          double imag = a.im + b.im;
80          return new Complex(real, imag);
81      }
82
83      // sample client for testing
84      public static void main(String[] args) {
85          Complex a = new Complex(5.0, 6.0);
86          Complex b = new Complex(-3.0, 4.0);
87
88          System.out.println("a            = " + a);
89          System.out.println("b            = " + b);
90          System.out.println("a + b        = " + Complex.plus(a, b));
91          System.out.println("(a / b) * b  = " + a.divides(b).times(b));
92      }
93
94  }
```

Κατασκευάζοντας μία κλάση με την ASM

```
1   public class Complex {
2
3       private double re;
4       private double im;
5
6       public Complex(double re, double im) {
7           this.re = re;
8           this.im = im;
9       }
10
11  }
```

```
1   // create class
2   ClassNode cn = new ClassNode();
3   cn.access = Opcodes.ACC_PUBLIC;
4   cn.version = Opcodes.V1_5;
5   cn.name = "Complex";
6   cn.sourceFile = "Complex.java";
7   cn.superName = "java/lang/Object";
8
9   // add fields
10  cn.fields.add(new FieldNode(Opcodes.ACC_PRIVATE, "re", "D", null, null));
11  cn.fields.add(new FieldNode(Opcodes.ACC_PRIVATE, "im", "D", null, null));
12
13  // create constructor
14  MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC, "<init>", "(DD)V", null, null);
15  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
16  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "java/lang/Object", "<init>", "()V"));
17  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
18  mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 1));
19  mn.instructions.add(new FieldInsnNode(Opcodes.PUTFIELD, "Complex", "re", "D"));
20  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
21  mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 3));
22  mn.instructions.add(new FieldInsnNode(Opcodes.PUTFIELD, "Complex", "im", "D"));
23  mn.instructions.add(new InsnNode(Opcodes.RETURN));
24  mn.maxStack = 3;
25  mn.maxLocals = 5;
26  cn.methods.add(mn);
```

Κατασκευάζοντας μία κλάση με την ASM

```
1  public Complex plus(Complex b) {
2      double re = this.re + b.re;
3      double im = this.im + b.im;
4      return new Complex(re, im);
5  }
```

```
1  MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC, "plus", "(LComplex;)LComplex;", null, null);
2  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
3  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
4  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 1));
5  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
6  mn.instructions.add(new InsnNode(Opcodes.DADD));
7  mn.instructions.add(new VarInsnNode(Opcodes.DSTORE, 2));
8
9  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
10 mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
11 mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 1));
12 mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
13 mn.instructions.add(new InsnNode(Opcodes.DADD));
14 mn.instructions.add(new VarInsnNode(Opcodes.DSTORE, 4));
15
16 mn.instructions.add(new TypeInsnNode(Opcodes.NEW, "Complex"));
17 mn.instructions.add(new InsnNode(Opcodes.DUP));
18 mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 2));
19 mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 4));
20 mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "Complex", "<init>", "(DD)V"));
21
22 mn.instructions.add(new InsnNode(Opcodes.ARETURN));
23 mn.maxStack = 6;
24 mn.maxLocals = 6;
```

Κατασκευάζοντας μία κλάση με την ASM

```
1    public Complex minus(Complex b) {
2        double re = this.re − b.re;
3        double im = this.im − b.im;
4        return new Complex(re, im);
5    }
```

```
1    MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC, "minus", "(LComplex;)LComplex;", null, null);
2    mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
3    mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
4    mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 1));
5    mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
6    mn.instructions.add(new InsnNode(Opcodes.DADD));
7    mn.instructions.add(new VarInsnNode(Opcodes.DSTORE, 2));
8
9    mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
10   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
11   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 1));
12   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
13   mn.instructions.add(new InsnNode(Opcodes.DSUB));
14   mn.instructions.add(new VarInsnNode(Opcodes.DSTORE, 4));
15
16   mn.instructions.add(new TypeInsnNode(Opcodes.NEW, "Complex"));
17   mn.instructions.add(new InsnNode(Opcodes.DUP));
18   mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 2));
19   mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 4));
20   mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "Complex", "<init>", "(DD)V"));
21
22   mn.instructions.add(new InsnNode(Opcodes.ARETURN));
23   mn.maxStack = 6;
24   mn.maxLocals = 6;
```

Κατασκευάζοντας μία κλάση με την ASM

```
1   // return a new Complex object whose value is (this * b)
2   public Complex times(Complex b) {
3       double real = this.re * b.re - this.im * b.im;
4       double imag = this.re * b.im + this.im * b.re;
5       return new Complex(real, imag);
6   }
```

```
1   MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC, "times", "(LComplex;)LComplex;", null, null);
2   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
3   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
4   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 1));
5   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
6   mn.instructions.add(new InsnNode(Opcodes.DMUL));
7   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
8   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
9   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 1));
10  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
11  mn.instructions.add(new InsnNode(Opcodes.DMUL));
12  mn.instructions.add(new InsnNode(Opcodes.DSUB));
13  mn.instructions.add(new VarInsnNode(Opcodes.DSTORE, 2));
14  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
15  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
16  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 1));
17  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
18  mn.instructions.add(new InsnNode(Opcodes.DMUL));
19  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
20  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
21  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 1));
22  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
23  mn.instructions.add(new InsnNode(Opcodes.DMUL));
24  mn.instructions.add(new InsnNode(Opcodes.DADD));
25  mn.instructions.add(new VarInsnNode(Opcodes.DSTORE, 4));
26  mn.instructions.add(new TypeInsnNode(Opcodes.NEW, "Complex"));
27  mn.instructions.add(new InsnNode(Opcodes.DUP));
28  mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 2));
29  mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 4));
30  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "Complex", "<init>", "(DD)V"));
31  mn.instructions.add(new InsnNode(Opcodes.ARETURN));
32  mn.maxStack = 6;
```

Κατασκευάζοντας μία κλάση με την ASM

```
1    // scalar multiplication
2    // return a new object whose value is (this * alpha)
3    public Complex times(double alpha) {
4        return new Complex(alpha * re, alpha * im);
5    }
```

```
1    MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC, "times", "(D)LComplex;", null, null);
2    mn.instructions.add(new TypeInsnNode(Opcodes.NEW, "Complex"));
3    mn.instructions.add(new InsnNode(Opcodes.DUP));
4    mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 1));
5    mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
6    mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
7    mn.instructions.add(new InsnNode(Opcodes.DMUL));
8    mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 1));
9    mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
10   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
11   mn.instructions.add(new InsnNode(Opcodes.DMUL));
12   mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "Complex", "<init>", "(DD)V"));
13   mn.instructions.add(new InsnNode(Opcodes.ARETURN));
14   mn.maxStack = 8;
15   mn.maxLocals = 3;
```

Κατασκευάζοντας μία κλάση με την ASM

```
1  public double abs() {
2      // Math.sqrt(re*re + im*im)
3      return Math.hypot(re, im);
4  }
```

```
1  MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC, "abs", "()D", null, null);
2  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
3  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
4  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
5  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
6  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESTATIC, "java/lang/Math", "hypot", "(DD)D"));
7  mn.instructions.add(new InsnNode(Opcodes.DRETURN));
8  mn.maxStack = 4;
9  mn.maxLocals = 1;
```

Κατασκευάζοντας μία κλάση με την ASM

```
1    public double re() {
2        return re;
3    }
```

```
1    MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC, "re", "()D", null, null);
2    mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
3    mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
4    mn.instructions.add(new InsnNode(Opcodes.DRETURN));
5    mn.maxStack = 2;
6    mn.maxLocals = 1;
```

Κατασκευάζοντας μία κλάση με την ASM

```
1    public double im() {
2        return im;
3    }
```

```
1    MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC, "im", "()D", null, null);
2    mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
3    mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
4    mn.instructions.add(new InsnNode(Opcodes.DRETURN));
5    mn.maxStack = 2;
6    mn.maxLocals = 1;
```

Κατασκευάζοντας μία κλάση με την ASM

```
1   // return a new Complex object whose value is the reciprocal of this
2   public Complex reciprocal() {
3       double scale = re * re + im * im;
4       return new Complex(re / scale, -im / scale);
5   }
```

```
1   MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC, "reciprocal", "()LComplex;", null, null);
2   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
3   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
4   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
5   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
6   mn.instructions.add(new InsnNode(Opcodes.DMUL));
7   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
8   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
9   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
10  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
11  mn.instructions.add(new InsnNode(Opcodes.DMUL));
12  mn.instructions.add(new InsnNode(Opcodes.DADD));
13  mn.instructions.add(new VarInsnNode(Opcodes.DSTORE, 1));
14  mn.instructions.add(new TypeInsnNode(Opcodes.NEW, "Complex"));
15  mn.instructions.add(new InsnNode(Opcodes.DUP));
16  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
17  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
18  mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 1));
19  mn.instructions.add(new InsnNode(Opcodes.DDIV));
20  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
21  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
22  mn.instructions.add(new InsnNode(Opcodes.DNEG));
23  mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 1));
24  mn.instructions.add(new InsnNode(Opcodes.DDIV));
25  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "Complex", "<init>", "(DD)V"));
26  mn.instructions.add(new InsnNode(Opcodes.ARETURN));
27  mn.maxStack = 8;
28  mn.maxLocals = 3;
```

Κατασκευάζοντας μία κλάση με την ASM

```
1   // return a / b
2   public Complex divides(Complex b) {
3       return this.times(b.reciprocal());
4   }
```

```
1   new MethodNode(Opcodes.ACC_PUBLIC, "divides", "(LComplex;)LComplex;", null, null);
2   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
3   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 1));
4   mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "Complex", "reciprocal", "()LComplex;"));
5   mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "Complex", "times", "(LComplex;)LComplex;"));
6   mn.instructions.add(new InsnNode(Opcodes.ARETURN));
7   mn.maxStack = 2;
8   mn.maxLocals = 2;
```

Κατασκευάζοντας μία κλάση με την ASM

```
1   // a static version of plus
2   public static Complex plus(Complex a, Complex b) {
3       double real = a.re + b.re;
4       double imag = a.im + b.im;
5       return new Complex(real, imag);
6   }
```

```
1   MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC + Opcodes.ACC_STATIC, "plus", "(LComplex;LComplex;)LComplex;", null, null);
2   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
3   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
4   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 1));
5   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
6   mn.instructions.add(new InsnNode(Opcodes.DADD));
7   mn.instructions.add(new VarInsnNode(Opcodes.DSTORE, 2));
8   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
9   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
10  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 1));
11  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
12  mn.instructions.add(new InsnNode(Opcodes.DADD));
13  mn.instructions.add(new VarInsnNode(Opcodes.DSTORE, 4));
14  mn.instructions.add(new TypeInsnNode(Opcodes.NEW, "Complex"));
15  mn.instructions.add(new InsnNode(Opcodes.DUP));
16  mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 2));
17  mn.instructions.add(new VarInsnNode(Opcodes.DLOAD, 4));
18  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "Complex", "<init>", "(DD)V"));
19  mn.instructions.add(new InsnNode(Opcodes.ARETURN));
20  mn.maxStack = 4;
21  mn.maxLocals = 6;
```

Κατασκευάζοντας μία κλάση με την ASM

```
1   public String toString() {
2       if (im == 0) {
3           return re + "";
4       }
5       if (re == 0) {
6           return im + "i";
7       }
8       if (im < 0) {
9           return re + " - " + (-im) + "i";
10      }
11      return re + " + " + im + "i";
12  }
```

```
1   MethodNode mn = new MethodNode(Opcodes.ACC_PUBLIC, "toString", "()Ljava/lang/String;", null, null);
2   mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
3   mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
4   mn.instructions.add(new InsnNode(Opcodes.DCONST_0));
5   mn.instructions.add(new InsnNode(Opcodes.DCMPL));
6   LabelNode l0 = new LabelNode(new Label());
7   mn.instructions.add(new JumpInsnNode(Opcodes.IFNE, l0));
8   mn.instructions.add(new TypeInsnNode(Opcodes.NEW, "java/lang/StringBuilder"));
9   mn.instructions.add(new InsnNode(Opcodes.DUP));
10  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "java/lang/StringBuilder", "<init>", "()V"));
11  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
12  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
13  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "append", "(D)Ljava/lang/StringBuilder;"));
14  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "toString", "()Ljava/lang/String;"));
15  mn.instructions.add(new InsnNode(Opcodes.ARETURN));
16
17  mn.instructions.add(l0);
```

Κατασκευάζοντας μία κλάση με την ASM

```
1   public String toString() {
2       if (im == 0) {
3           return re + "";
4       }
5       if (re == 0) {
6           return im + "i";
7       }
8       if (im < 0) {
9           return re + " - " + (-im) + "i";
10      }
11      return re + " + " + im + "i";
12  }
```

```
18  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
19  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
20  mn.instructions.add(new InsnNode(Opcodes.DCONST_0));
21  mn.instructions.add(new InsnNode(Opcodes.DCMPL));
22  LabelNode l1 = new LabelNode(new Label());
23  mn.instructions.add(new JumpInsnNode(Opcodes.IFNE, l1));
24  mn.instructions.add(new TypeInsnNode(Opcodes.NEW, "java/lang/StringBuilder"));
25  mn.instructions.add(new InsnNode(Opcodes.DUP));
26  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "java/lang/StringBuilder", "<init>", "()V"));
27  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
28  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
29  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "append", "(D)Ljava/lang/StringBuilder;"));
30  mn.instructions.add(new LdcInsnNode("i"));
31  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "append", "(Ljava/lang/String;)Ljava/lang/StringBuild
32  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "toString", "()Ljava/lang/String;"));
33  mn.instructions.add(new InsnNode(Opcodes.ARETURN));
34
35  mn.instructions.add(l1);
```

Κατασκευάζοντας μία κλάση με την ASM

```
1   public String toString() {
2       if (im == 0) {
3           return re + "";
4       }
5       if (re == 0) {
6           return im + "i";
7       }
8       if (im < 0) {
9           return re + " - " + (-im) + "i";
10      }
11      return re + " + " + im + "i";
12  }
```

```
36  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
37  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
38  mn.instructions.add(new InsnNode(Opcodes.DCONST_0));
39  mn.instructions.add(new InsnNode(Opcodes.DCMPL));
40  LabelNode l2 = new LabelNode(new Label());
41  mn.instructions.add(new JumpInsnNode(Opcodes.IFGE, l2));
42  mn.instructions.add(new TypeInsnNode(Opcodes.NEW, "java/lang/StringBuilder"));
43  mn.instructions.add(new InsnNode(Opcodes.DUP));
44  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "java/lang/StringBuilder", "<init>", "()V"));
45  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
46  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
47  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "append", "(D)Ljava/lang/StringBuilder;"));
48  mn.instructions.add(new LdcInsnNode(" - "));
49  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "append", "(Ljava/lang/String;)Ljava/lang/StringBuilde
50  mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
51  mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
52  mn.instructions.add(new InsnNode(Opcodes.DNEG));
53  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "append", "(D)Ljava/lang/StringBuilder;"));
54  mn.instructions.add(new LdcInsnNode("i"));
55  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "append", "(Ljava/lang/String;)Ljava/lang/StringBuilde
56  mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "toString", "()Ljava/lang/String;"));
57  mn.instructions.add(new InsnNode(Opcodes.ARETURN));
58
59  mn.instructions.add(12);
```

Κατασκευάζοντας μία κλάση με την ASM

```
60    mn.instructions.add(new TypeInsnNode(Opcodes.NEW, "java/lang/StringBuilder"));
61    mn.instructions.add(new InsnNode(Opcodes.DUP));
62    mn.instructions.add(new MethodInsnNode(Opcodes.INVOKESPECIAL, "java/lang/StringBuilder", "<init>", "()V"));
63    mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
64    mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "re", "D"));
65    mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "append", "(D)Ljava/lang/StringBuilder;"));
66    mn.instructions.add(new LdcInsnNode(" + "));
67    mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "append", "(Ljava/lang/String;)Ljava/lang/StringBuilde
68    mn.instructions.add(new VarInsnNode(Opcodes.ALOAD, 0));
69    mn.instructions.add(new FieldInsnNode(Opcodes.GETFIELD, "Complex", "im", "D"));
70    mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "append", "(D)Ljava/lang/StringBuilder;"));
71    mn.instructions.add(new LdcInsnNode("i"));
72    mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "append", "(Ljava/lang/String;)Ljava/lang/StringBuilde
73    mn.instructions.add(new MethodInsnNode(Opcodes.INVOKEVIRTUAL, "java/lang/StringBuilder", "toString", "()Ljava/lang/String;"));
74    mn.instructions.add(new InsnNode(Opcodes.ARETURN));
75
76    mn.maxStack = 4;
77    mn.maxLocals = 1;
```