Καρακασίδης Αλέξανδρος
Καστίδου Γεωργία
Παπαφώτη Μαρία
Πέτσιος Κων/νος – Στέφανος
Σαλτέας Καλογεράς Παναγιώτης

Threads in Java

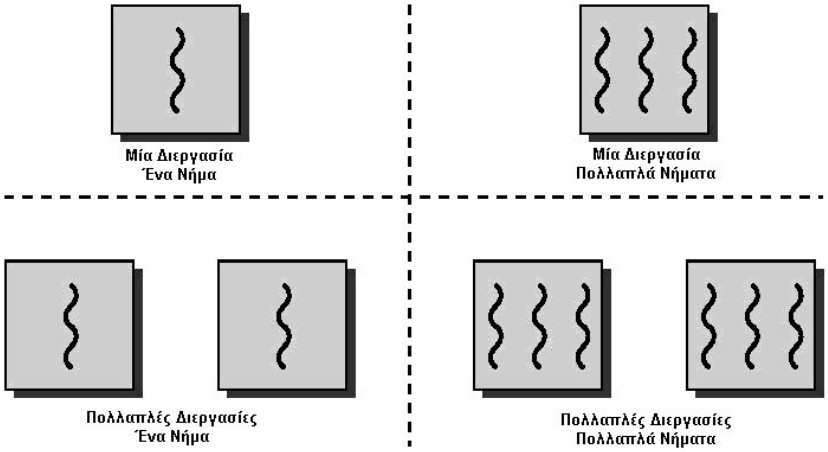
ΝΗΜΑΤΑ ΣΤΗ JAVA

1. Εισαγωγή – Τι είναι Νήμα;

Κάθε νήμα εκτέλεσης είναι ουσιαστικά μια ανεξάρτητη ροή του ελέγχου εκτέλεσης, μέσα στο πρόγραμμα. Η έννοια του νήματος (thread) είναι παρόμοια με αυτήν της διεργασίας (process), με τη διαφορά ότι τα διάφορα νήματα μοιράζονται τον ίδιο χώρο διευθύνσεων, κάτι που δε συμβαίνει συνήθως με τις διεργασίες. Με άλλα λόγια δυο διαφορετικά νήματα μέσα σε ένα πρόγραμμα έχουν τη δυνατότητα πρόσβασης σε κάποιο αντικείμενο του προγράμματος ενώ δυο διαφορετικές διεργασίες συνήθως βλέπουν διαφορετικά αντίγραφα του ίδιου αντικειμένου.

2. Σχέση Νημάτων και Διεργασιών

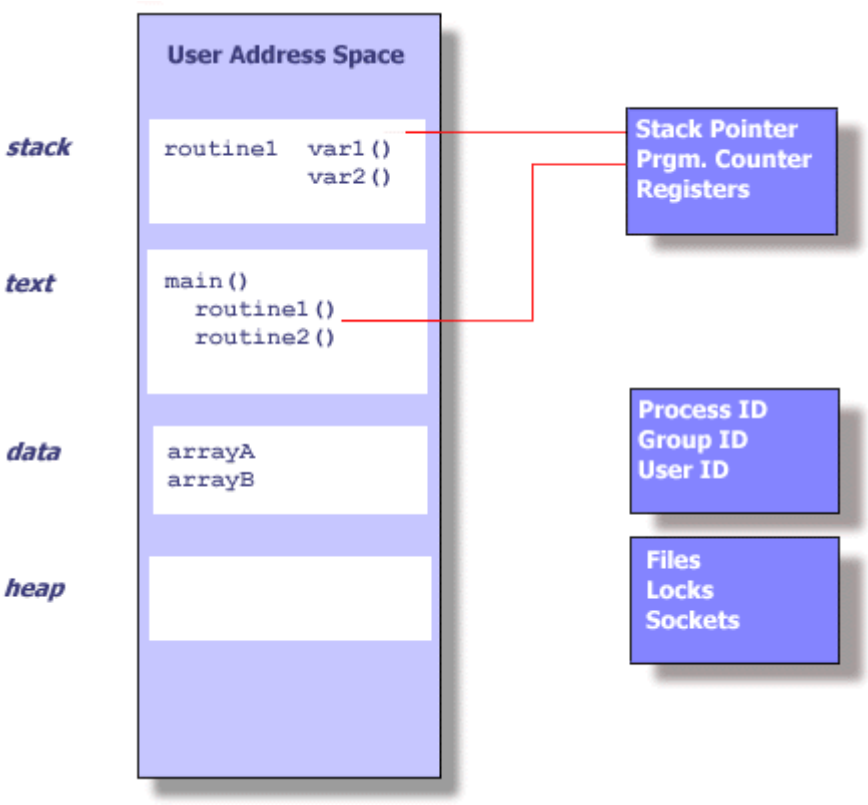
Για την καλύτερη κατανόηση της τεχνολογίας των νημάτων, θα ήταν χρήσιμο να διευκρινιστεί η σχέση μεταξύ μίας διεργασίας και ενός νήματος. Ένα νήμα ανήκει σε μία διεργασία, και χρησιμοποιεί τους πόρους αυτής της διεργασίας. Το νήμα όμως κατέχει τη δική του, αυτόνομη ροή ελέγχου. Σε μία διεργασία μπορούν να ανήκουν πολλά νήματα. Μία καλύτερη περιγραφή προκύπτει αν θεωρήσουμε το νήμα σαν μία διαδικασία η οποία να μπορεί να εκτελείται αυτόνομα σε μία διεργασία.



Σχήμα 1: Διεργασίες και Νήματα

Μία διεργασία δημιουργείται από το λειτουργικό σύστημα. Οι διεργασίες περιέχουν πληροφορίες για τους πόρους του προγράμματος, καθώς και για την κατάσταση εκτέλεσης του προγράμματος, συμπεριλαμβανομένων:

- Ταυτότητα διεργασίας, ταυτότητα ομάδας διεργασίας, ταυτότητα χρήστη κ.τ.λ.
- Περιβάλλον εκτέλεσης
- Κατάλογος εργασίας
- Εντολές του προγράμματος
- Καταχωρητές
- Στοίβα
- Κοινός χώρος διευθύνσεων, σωρός δεδομένων και μνήμης
- Δείκτες αρχείων
- Διαμοιραζόμενες βιβλιοθήκες
- Εργαλεία διαδιεργασιακής επικοινωνίας (όπως ουρές μηνυμάτων, σωληνώσεις, σημαφόροι, ή κοινή μνήμη)

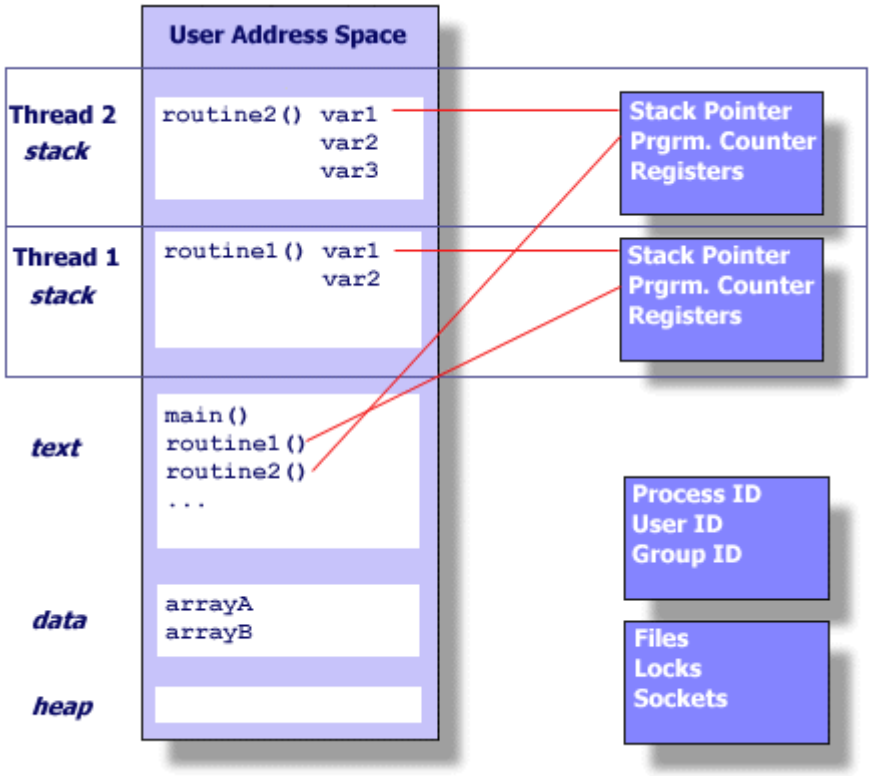


Σχήμα 2: Χαρακτηριστικά μία διεργασίας

Τα νήματα χρησιμοποιούν και υπάρχουν μέσα στους πόρους της διεργασίας, ενώ μπορούν να προγραμματιστούν από το λειτουργικό σύστημα και να εκτελούνται ως αυτόνομες οντότητες εντός της διεργασίας στην οποία ανήκουν.

Ένα νήμα μπορεί να έχει ανεξάρτητη ροή ελέγχου, και να είναι προγραμματίσιμο γιατί διατηρεί:

- Στοίβα
- Ιδιότητες χρονοπρογραμματισμού (όπως προτεραιότητα)
- Σύνολο σημάτων σε αναμονή και σε διακοπή
- Δεδομένα των νημάτων.



Σχήμα 3: Χαρακτηριστικά ενός νήματος

Σε μία διεργασία μπορεί να ανήκουν πολλά νήματα, τα οποία μοιράζονται τους πόρους της και εκτελούνται στον ίδιο χώρο διευθύνσεων. Σε ένα πολυνηματικό πρόγραμμα (multi-threaded program) υπάρχουν ανά πάσα στιγμή πολλαπλά σημεία εκτέλεσης.

Ο διαμοιρασμός των πόρων από τα νήματα έχει τα εξής αποτελέσματα:

- Οι αλλαγές που πραγματοποιούνται από ένα νήμα στους διαμοιραζόμενους πόρους (όπως, για παράδειγμα, το κλείσιμο ενός αρχείου), είναι ορατές από όλα τα άλλα νήματα.
- Δύο δείκτες που έχουν την ίδια τιμή αναφέρονται στο ίδιο δεδομένο.
- Είναι δυνατή η ανάγνωση και η εγγραφή στην ίδια θέση μνήμης.

Τέλος, οι ιδιότητες αυτές των νημάτων παρέχουν διαφορετικές ιδιότητες, ανάλογα με το περιβάλλον στο οποίο χρησιμοποιούνται. Σε ένα μονοεπεξεργαστικό σύστημα, οι πολυνηματικές διεργασίες παρέχουν ταυτόχρονη εκτέλεση. Σε ένα πολυεπεξεργαστικό περιβάλλον, μία πολυνηματική διεργασία προσφέρει παραλληλισμό.

3. Δημιουργία Νημάτων στην JAVA

Όταν αρχίζει η εκτέλεση ενός Java προγράμματος, μπορεί κανείς να φαντασθεί ότι υπάρχει μόνο ένα νήμα εκτέλεσης. Η παρατήρηση αυτή δεν είναι ακριβής αφού πρώτον μαζί με κάθε νήμα εκτέλεσης ενός Java προγράμματος τρέχει τουλάχιστον και ο συλλέκτης σκουπιδιών και δεύτερον στην περίπτωση των Java applets, ο αριθμός των νημάτων εκτέλεσης αυξάνεται κατά πολύ επειδή δημιουργεί νέα νήματα το ίδιο το σύστημα όποτε το κρίνει σκόπιμο. Για να δημιουργήσει όμως ο χρήστης ένα νέο νήμα, πρέπει να δημιουργήσει ένα νέο αντικείμενο, που να προέρχεται από την κλάση Thread, που ορίζεται στο πακέτο java.lang, ή από κάποια υποκλάση της. Το νέο αυτό αντικείμενο παριστάνει μια νέα εργασία που πρέπει να αρχίσει να εκτελείται παράλληλα με την παλιά. Υπάρχουν δύο τρόποι για την υλοποίηση νέων νημάτων εκτέλεσης. Παρακάτω περιγράφονται και οι δύο αυτοί τρόποι. Σε κάθε περίπτωση ο κώδικάς που υλοποιεί τη νέα εργασία τοποθετείται σε μια μέθοδο με όνομα run.

- *Πρώτος τρόπος για την υλοποίηση νέων νημάτων εκτέλεσης.* Η νέα εργασία υλοποιείται από μια μέθοδο με το όνομα run η οποία δηλώνεται ως μέθοδος μιας νέας κλάσης που δημιουργείται και υλοποιεί τη διαπροσωπεία Runnable. Στο Σχήμα 4α φαίνεται ένα παράδειγμα μιας κλάσης που υλοποιεί την έπ' άπειρον εκτύπωση στην οθόνη της τιμής ενός μετρητή.

```
class CountingJob implements Runnable {
      private int counter;
      public CountingJob (int c) { counter = c; }
      public void run( ) {
            while(true)
                  System.out.println(counter++);
      }
}
```

(α)

```
CountingJob job = new CountingJob(1);
Thread t = new Thread (job);
t.start();
```

(β)

Σχήμα 4: Παράδειγμα δημιουργίας νέου νήματος εκτέλεσης

Στη συνέχεια, για να αρχίσει η εκτέλεση της εργασίας, απαιτείται η δημιουργία ενός αντικειμένου τύπου Thread, περνώντας σον κατασκευαστή του ένα αντικείμενο της νέας εργασίας ως παράμετρο. Η έναρξη της εκτέλεσης πραγματοποιείται καλώντας τη μέθοδο start του αντικειμένου

Thread. Μετά την κλήση της start η εκτέλεση της νέας εργασίας συνεχιζεται είτε ώσπου να τελειώσει η εκτέλεση της μεθόδου Run, είτε μέχρι να κληθεί η μέθοδος stop του αντικειμένου Thread. Παράδειγμα του μέρους αυτού του κώδικα φαίνεται στο σχήμα 4β. Σημειώνεται εδώ ότι είναι επίσης δυνατόν το αντικείμενο τύπου Thread να αποτελεί τμήμα της κλάσης CountingJob και η εκτέλεση της νέας εργασίας να αρχίζει στον κατασκευαστή αυτής της κλάσης.

- Δεύτερος τρόπος για την υλοποίηση νέων νημάτων εκτέλεσης. Η νέα εργασία υλοποιείται από μια μέθοδο run που ορίζεται από μια νέα υποκλάση της Thread. Με βάση αυτόν τον τρόπο το παράδειγμα του Σχήματος 4 θα είχε τη μορφή που φαίνεται στο Σχήμα 5.

```
class CountingJob extends Thread {
      private int counter;
      public CountingJob (int c)  {
            counter = c;
            start();
      }

      public void run( ) {
            while(true)
                  System.out.println(counter++);
      }
}
```

Σχήμα 5: Παράδειγμα δημιουργίας νέου νήματος εκτέλεσης ($2^{ος}$ τρόπος)

4. Χειρισμός Νημάτων στην JAVA

Εκτός από τις μεθόδους start και stop της κλάσης Thread που χρησιμοποιούνται για την εκκίνηση και τον τερματισμό της εκτέλεσης ενός νήματος η κλάση παρέχει κι άλλες μεθόδους για τον περαιτέρω χειρισμό των νημάτων. Τέτοιες είναι οι:

- suspend και resume που χρησιμοποιούνται για την προσωρινή παύση της εκτέλεσης ενός νήματος και την επανεκκίνηση της αντίστοιχα. Οι μέθοδοι αυτές σε αντίθεση με τις start και stop μπορούν να κληθούν περισσότερες από μια φορές.
- sleep(n) χρησιμοποιείται για την καθυστέρηση ενός νήματος εκτέλεσης κατά n milliseconds.

- currentThread πρόκειται για μια στατική μέθοδο η οποία επιστρέφει το τρέχον νήμα εκτέλεσης και είναι χρήσιμή για την πρόσβαση στο αρχικό νήμα.
- setDaemon Η μέθοδος αυτή παίρνει ως παράμετρο μια τιμή Boolean. Αν η τιμή της παραμέτρου είναι true τότε το νήμα χαρακτηρίζεται ως δαίμων. Οι «δαίμονες» είναι νήματα που καταστρέφονται αυτόματα μετά τον τερματισμό όλων των άλλων νημάτων.
- setPriority Αυτή η μέθοδος χρησιμοποιείται για τον καθορισμό της προτεραιότητας εκτέλεσης του νήματος. Η προτεραιότητα είναι μια αριθμητική τιμή : όσο μεγαλύτερη είναι αυτή τόσο περισσότερος χρόνος αφιερώνεται από τον υπολογιστή για την εκτέλεση του νήματος. Με άλλα λόγια νήματα με μεγαλύτερη προτεραιότητα πριμοδοτούνται από το διερμηνέα της Java έναντι νημάτων με μικρότερη προτεραιότητα.
- yield Αυτή η μέθοδος χρησιμοποιείται για να παραχωρήσει ένα νήμα το χρόνο υπολογισμού που του έχει δοθεί.

5. Συγχρονισμός νημάτων

Στις περισσότερες περιπτώσεις, όταν πολλά νήματα εκτέλεσης διαχειρίζονται το ίδιο σύνολο αντικειμένων, είναι επιθυμητό να υπάρχει συγχρονισμός στις εργασίες που επιτελούνται. Αυτό είναι απαραίτητο, γιατί διαφορετικά τα αποτελέσματα ενδέχεται να είναι καταστροφικά. Αν, για παράδειγμα, ένα νήμα εκτέλεσης διαβάζει μια μεταβλητή ενός αντικειμένου την ίδια στιγμή που ένα άλλο νήμα εκχωρεί μια τιμή σε αυτή, το αποτέλεσμα και των δύο ενεργειών είναι εντελώς απρόβλεπτο. Το πρόβλημα του συγχρονισμού παράλληλων διεργασιών είναι εξαιρετικά δύσκολο, αφού εκτός από το να παρέχει τη δυνατότητα αποκλειστικής χρήσης στα αντικείμενα, ένας μηχανισμός συγχρονισμού πρέπει να εξασφαλίζει την αποφυγή αδιεξόδων.

Η Java παρέχει έναν αρκετά απλό και εύχρηστο μηχανισμό για το συγχρονισμό των νημάτων εκτέλεσης. Ο μηχανισμός αυτός βασίζεται στην έννοια του *κλειδώματος* (lock). Κάθε αντικείμενο της Java μπορεί να θεωρηθεί ότι διαθέτει ένα κλειδί. Το ίδιο συμβαίνει και για κάθε κλάση. Προκειμένου να επιτευχθεί αποκλειστική πρόσβαση σε ένα σύνολο μεθόδων ενός αντικειμένου, αυτές δηλώνονται ως synchronized (συγχρονισμένες). Στη συνέχεια για να κληθεί μια συγχρονισμένη μέθοδος ενός

αντικειμένου, πρέπει να αποκτηθεί το κλειδί του αντικειμένου από το νήμα εκτέλεσης που πραγματοποιεί την κλήση. Το κλειδί επιστρέφεται μετά την εκτέλεση της μεθόδου.

Ας θεωρήσουμε για παράδειγμα την ακόλουθη κλάση SafeVariable. Η κλάση αυτή ορίζει δύο συγχρονισμένες μεθόδους get και put, που χρησιμοποιούνται για την ανάκτηση και την αποθήκευση μιας ακέραιας τιμής από ένα αντικείμενο (Σχήμα 6).

```
public class SafeVariable
{
        private int value;

        public synchronized int get() { return value; }
        public synchronized void put(int v) { value = v; }
}
```

Σχήμα 6: Παράδειγμα κλάσης SafeVariable

Με τη χρήση των μεθόδων αυτών εξασφαλίζονται ότι για κάθε αντικείμενο αυτής της κλάσης, το πολύ μια μέθοδος θα εκτελείται για κάθε χρονική στιγμή. Τα κλειδιά των κλάσεων χρησιμοποιούνται για στατικές συγχρονισμένες μεθόδους, κατά τρόπο παρόμοιο με τα κλειδιά των αντικειμένων.

Εκτός από τη δήλωση συγχρονισμένων μεθόδων, είναι δυνατός ο συγχρονισμός αυθαίρετων τμημάτων κώδικα, με χρήση και πάλι της λέξης synchronized. Για παράδειγμα, στο παρακάτω τμήμα κώδικα, προκειμένου να εκτελεστεί το εσωτερικό της εντολής πρέπει να αποκτηθεί το κλειδί του αντικειμένου obj. Αυτό το τμήμα κώδικα μπορεί όμως να ανήκει σε οποιαδήποτε μέθοδο, όχι απαραίτητα του αντικειμένου αυτού (σχήμα 7).

```
synchronized  (obj ) {
        // manipulate obj in some way…
}
```

Σχήμα 7: Χρήση μεθόδου synchronized

Εκτός από το συγχρονισμό μεθόδων με χρήση της λέξης synchronized, η Java παρέχει τη δυνατότητα συγχρονισμού με τις μεθόδους wait και notify της κλάσης Object. Οι μέθοδοι αυτές ορίζονται για κάθε αντικείμενο, καθώς όλες οι κλάσεις

κληρονομούν την Object. Οι κλήσεις σε αυτές τις μεθόδους πραγματοποιούνται σχεδόν πάντα μέσα από συγχρονισμένες μεθόδους ή τμήματα κώδικα.
Με την κλήση της μεθόδου wait για κάποιο αντικείμενο, το τρέχον νήμα εκτέλεσης επιστρέφει το κλειδί του αντικειμένου. Αν το νήμα βρίσκεται σε μια συγχρονισμένη μέθοδο αυτού του αντικειμένου, η επιστροφή του κλειδιού έχει ως αποτέλεσμα να σταματήσει προσωρινά η εκτέλεση του νήματος. Αυτό είναι επιθυμητό στην περίπτωση που η εκτέλεση της εργασίας δεν είναι δυνατό να ολοκληρωθεί λόγω έλλειψης πληροφοριών, για τις οποίες πρέπει το τρέχον νήμα να περιμένει.

Για να συνεχίσει η εκτέλεση θα πρέπει να συμβούν δύο πράγματα : αφενός να, ενημερωθεί το νήμα εκτέλεσης ότι μπορεί να συνεχίσει, μέσω της κλήσης στη μέθοδο notify του αντικειμένου από ένα άλλο νήμα, και αφετέρου να αποκτηθεί και πάλι το κλειδί. Για κάθε κλήση της μεθόδου notify ενός αντικειμένου, ο διερμηνέας της Java ενημερώνει μόνο μια μέθοδο που έχει εκτελέσει τη μέθοδο wait γι’ αυτό το αντικείμενο. Αν πρέπει να ενημερωθούν όλες οι μέθοδοι που βρίσκονται σε αναμονή, μπορεί να χρησιμοποιηθεί η μέθοδο notifyAll.

Αναφορές

[1] Αλέξανδρος Καρακασίδης, «Πολυνηματικός Προσομοιωτής Ασύρματου τοπικού δικτύου IEEE 802.11», Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Μάρτιος 2002.

[2] Σοφοκλής Εφραιμίδης, Αντώνης Καβάρνος, Τάσος Κουτουμάνος, Βασίλης Παπαδήμος, Νίκος Παπασπύρου, Γιάννης Πατινιωτάκης, Κλειώ Σγουροπούλου, Εισαγωγή στη Γλώσσα Προγραμματισμού Java.

[3] Andrew S. Tanenbaum and Maarten Van Steen, "Distributed Systems: Principles and Paradigms", Prentice Hall, 2002