Νήματα και Συγχρονισμός

Παναγιώτης Αδαμίδης

Τμήμα Μηχανικών Πληροφορικής
Αλεξάνδρειο ΤΕΙ Θεσσαλονίκης

Περιεχόμενα

1. Διεργασίες και Νήματα (Processes and Threads) 2

1.1 Αναφορά στις Διεργασίες (Processes) 3

1.2. Αναφορά στα Νήματα (Threads) 3

2. Πολυνηματικός Προγραμματισμός (Mulithreaded Programming) 4

3. Δημιουργία και Διαχείριση Νημάτων 6

3.1. Περιγραφή μεθόδων της κλάσης Thread 11

3.2. Χρόνος ζωής ενός νήματος 13

4. Συγχρονισμός νημάτων 15

4.1 Συγχρονισμός τμήματος εντολών για πρόσβαση σε μεταβλητή ... 16

4.2 Συγχρονισμένες μέθοδοι 20

5. Ζωτικότητα (Liveness) 22

5.1. Αδιέξοδο (Deadlock) 22

5.2. Ενεργό αδιέξοδο (livelock) και Λιμοκτονία (Starvation) 24

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ • ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Οι χρήστες υπολογιστών θεωρούμε δεδομένο ότι τα συστήματά μας μπορούν να κάνουν πολλά πράγματα ταυτόχρονα. Για παράδειγμα να μπορούμε χρησιμοποιούμε τον επεξεργαστή κειμένου ενώ ταυτόχρονα άλλες εφαρμογές "κατεβάζουν" αρχεία, εκτυπώνουν, παίζουν μουσική κλπ. Ακόμη και από μία μόνο εφαρμογή προσδοκούμε να κάνει πολλά πράγματα ταυτόχρονα. Από έναν επεξεργαστή κειμένου προσδοκούμε να μπορεί να ανταποκριθεί στην είσοδο από το πληκτρολόγιο ή το ποντίκι ακόμη και αν εκείνη την ώρα ανανεώνει την οθόνη ή μορφοποιεί το κείμενο. Ταυτόχρονος προγραμματισμός (concurrent programming) ονομάζεται η μεθοδολογία προγραμματισμού που μας επιτρέπει να υλοποιήσουμε την ταυτόχρονη εκτέλεση διαφόρων εργασιών στην ίδια υπολογιστική συσκευή.

Η Java είναι σχεδιασμένη να υποστηρίζει ταυτόχρονο προγραμματισμό μέσω της ίδιας της γλώσσας και των βιβλιοθηκών της. Από την έκδοση 5 το API της Java διαθέτει υψηλού επιπέδου υποστήριξη για ταυτόχρονο προγραμματισμό.

Εδώ θα δούμε τις βασικές δυνατότητες που διαθέτει η Java για να υποστηρίξει ταυτόχρονο προγραμματισμό.

1. Διεργασίες και Νήματα (Processes and Threads)

Στον ταυτόχρονο προγραμματισμό (concurrent programming), υπάρχουν δύο βασικά στοιχεία εκτέλεσης: οι *διεργασίες* (*processes*) και τα νήματα (*threads*). Στη γλώσσα προγραμματισμού Java, ο ταυτόχρονος προγραμματισμός έχει να κάνει κυρίως με νήματα (threads) αν και η διεργασίες έχουν επίσης την αξία τους.

Ένα υπολογιστικό σύστημα έχει πολλές ενεργές διεργασίες και νήματα. Αυτό ισχύει ακόμη και για συστήματα τα οποία διαθέτουν μόνο μία μονάδα επεξεργασίας. Σε αυτή την περίπτωση ο χρόνος επεξεργασίας μοιράζεται μεταξύ των διεργασιών και των νημάτων κάνοντας χρήση της δυνατότητας του λειτουργικού συστήματος να αναθέτει τμήματα του χρόνου της μονάδας επεξεργασίας σε αυτές (time sharing ή time slicing).

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ • ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Σήμερα, τα υπολογιστικά συστήματα διαθέτουν πολλούς επεξεργαστές (multiple processors) ή επεξεργαστές με πολλούς πυρήνες επεξεργασίας (multiple execution cores). Αυτό εξασφαλίζει την δυνατότητα των συστημάτων να εκτελούν ταυτόχρονες διεργασίες και νήματα.

1.1 Αναφορά στις Διεργασίες (Processes)

Μία διεργασία διαθέτει το δικό της περιβάλλον εκτέλεσης. Γενικά, μία διεργασία διαθέτει ένα πλήρες, ιδιωτικό σύνολο βασικών πόρων εκτέλεσης και κυρίως το δικό της χώρο μνήμης.

Οι διεργασίες συχνά αντιμετωπίζονται ως συνώνυμο των προγραμμάτων ή των εφαρμογών. Όμως αυτό που βλέπουν οι χρήστες ως μία εφαρμογή συνήθως είναι ένα σύνολο συνεργαζόμενων διεργασιών. Για να διευκολυνθεί η επικοινωνία μεταξύ των διεργασιών, τα περισσότερα λειτουργικά συστήματα υποστηρίζουν την επικοινωνία μεταξύ των διεργασιών (*Inter Process Communication* - IPC), όπως διοχετεύσεις (pipes) και υποδοχές (sockets). Το IPC χρησιμοποιείται όχι μόνο για την επικοινωνία μεταξύ των διαδικασιών του ίδιου συστήματος αλλά και μεταξύ διαφορετικών συστημάτων

Οι περισσότερες υλοποιήσεις της εικονικής μηχανής της Java (Java Virtual Machine - JVM) λειτουργούν ως μία διεργασία. Μια εφαρμογή Java μπορεί να δημιουργήσει πρόσθετες διαδικασίες χρησιμοποιώντας ένα αντικείμενο ProcessBuilder. Εφαρμογές με περισσότερες διεργασίες είναι πέρα από τους στόχους αυτού του μαθήματος.

1.2. Αναφορά στα Νήματα (Threads)

Όπως οι διεργασίες (processes), έτσι και τα νήματα(threads) παρέχουν ένα περιβάλλον εκτέλεσης, αλλά η δημιουργία ενός νήματος απαιτεί λιγότερους πόρους από ότι η δημιουργία μιας διεργασίας.

Τα νήματα υπάρχουν μέσα στις διεργασίες. Κάθε διεργασία έχει τουλάχιστον ένα νήμα. Τα νήματα μοιράζονται τους πόρους της διεργασίας, συμπεριλαμβάνοντας την μνήμη και τα αρχεία. Αυτό αυξάνει την απόδοση και εξασφαλίζει

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ • ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

αποτελεσματική, αν και κάποιες φορές προβληματική, επικοινωνία μεταξύ των νημάτων.

2. Πολυνηματικός Προγραμματισμός (Mulithreaded Programming)

Κατά τον πολυνηματικό (multithreaded) προγραμματισμό έχουμε την ταυτόχρονη εκτέλεση πολλών νημάτων μέσα στο ίδιο πρόγραμμα. Μπορούμε να θεωρήσουμε ένα νήμα που εκτελείται ως μία CPU που εκτελεί το πρόγραμμα. Όταν έχουμε πολλά νήματα που εκτελούν το ίδιο πρόγραμμα, είναι σαν να έχουμε πολλές CPU που εκτελούν το ίδιο πρόγραμμα.

Ο πολυνηματισμός (multithreading) είναι ένας πολύ καλός τρόπος να βελτιώσουμε την εκτέλεση κάποιων προγραμμάτων.

Τα πλεονεκτήματα του πολυνηματισμού περιλαμβάνουν:

- καλύτερη χρήση των διαθέσιμων πόρων
- πιο απλά προγράμματα σε κάποιες περιπτώσεις
- καλύτερη απόκριση των προγραμμάτων

Για παράδειγμα έστω μια εφαρμογή η οποία διαβάζει δεδομένα από αρχεία που βρίσκονται στο δίσκο και μετά τα επεξεργάζεται. Ας πούμε ότι ο χρόνος ανάγνωσης είναι 5sec και ο χρόνος επεξεργασίας 2sec. Έτσι η εφαρμογή μας για να επεξεργαστεί δύο αρχεία θα χρειαστεί συνολικά 14sec (5sec διάβασμα 1ου αρχείου + 2sec επεξεργασία 1ου αρχείου + 5sec διάβασμα 2ου αρχείου + 2sec επεξεργασία 2ου αρχείου). Σε μια πολυνηματική εφαρμογή, την ώρα που θα διαβάζουμε το 2ο αρχείο, μπορούμε ταυτόχρονα να επεξεργαζόμαστε τα δεδομένα που έχουμε διαβάσει από το 1ο αρχείο. Έτσι ο συνολικός που θα χρειαστεί η εφαρμογή μας μειώνεται κατά 2sec.

Από πλευράς σχεδιασμού είναι πιο εύκολο να κάνουμε δύο νήματα καθένα από τα οποία θα διαβάζει από το δίσκο και μετά θα επεξεργάζεται τα δεδομένα, παρά να έχουμε μια εφαρμογή με ένα νήμα το οποίο θα πρέπει να παρακολουθεί τις διαδικασίες ανάγνωσης όλων των αρχείων. Μπορούμε να ξεκινήσουμε δύο νήματα καθένα από τα οποία περιμένει να ελευθερωθεί ο δίσκος μέχρι να τον χρησιμοποιήσει. Έτσι έχουμε καλύτερη και συνεχή χρήση του δίσκου αλλά και καλύτερη χρήση της CPU. Επίσης όσο αυξάνεται το πλήθος των αρχείων που έχουμε

να διαχειριστούμε, τόσο αυξάνει και η πολυπλοκότητα της εφαρμογής, ενώ η πολυνηματική προσέγγιση παραμένει η ίδια.

Η βελτίωση της απόκρισης μιας εφαρμογής είναι ένας ακόμη λόγος για να μετατρέψουμε μια εφαρμογή ενός νήματος σε πολυνηματική. Αν για παράδειγμα πατήσουμε ένα κουμπί που αρχίζει μια χρονοβόρα εργασία τότε το νήμα που την εκτελεί πρέπει να τελειώσει αυτή την εργασία πριν ανανεώσει τα παράθυρα, τα κουμπιά κλπ. και έτσι η εφαρμογή μοιάζει να μην αποκρίνεται όσο η εργασία εκτελείται. Αντιθέτως σε μια πολυνηματική εφαρμογή, αυτή η χρονοβόρα εργασία μπορεί να ανατεθεί σε ένα νήμα εργασίας. Ενώ το νήμα εργασίας (worker thread) είναι απασχολημένο, το νήμα του παράθυρου (window thread) είναι ελεύθερο να αποκριθεί στις απαιτήσεις του χρήστη.

Δεν έχει όμως μόνο πλεονεκτήματα ο πολυνηματικός προγραμματισμός. Στους σύγχρονους υπολογιστές που οι επεξεργαστές τους διαθέτουν πολλούς πυρήνες ή διαθέτουν πολλούς επεξεργαστές, είναι δυνατό να υπάρξουν λάθη τα οποία δεν υπάρχουν σε εφαρμογές με ένα επεξεργαστή όπου τα νήματα δεν θα εκτελεστούν παράλληλα.

Σε σύγχρονες πολυνηματικές (multithreaded) εφαρμογές, όπου τα νήματα εκτελούνται παράλληλα μπορεί να υπάρξουν λάθη. Θέλει προσοχή προκειμένου να μην δημιουργηθούν προβλήματα τα οποία σε μια μονο-νηματική εφαρμογή δεν υπάρχουν. Τα νήματα εκτελούνται στο ίδιο πρόγραμμα και έτσι διαβάζουν από, και γράφουν στη μνήμη ταυτόχρονα.

Για παράδειγμα τι θα συμβεί εάν ένα νήμα διαβάζει μια θέση μνήμης ενώ ένα άλλο νήμα γράφει σε αυτή; Ποια τιμή θα διαβάσει το πρώτο νήμα; Την παλιά τιμή, τη νέα ή μία μείξη αυτών; Ή, αν δύο νήματα γράφουν στην ίδια θέση μνήμης, ποια τιμή θα μείνει όταν τελειώσουν; Χωρίς τα κατάλληλα μέτρα προφύλαξης δεν ξέρουμε τι θα συμβεί. Η συμπεριφορά δεν είναι προβλέψιμη. Το αποτέλεσμα μπορεί να είναι διαφορετικό κάθε φορά που εκτελείται η εφαρμογή. Γι' αυτό ως προγραμματιστές είναι απαραίτητο να γνωρίζουμε τα μέτρα που πρέπει να πάρουμε για να ελέγχουμε

την πρόσβαση σε διαμοιραζόμενους πόρους όπως η μνήμη, τα αρχεία, οι βάσεις δεδομένων κλπ.

Γι' αυτό δεν πρέπει να δημιουργούμε νήματα εκεί που δεν μας χρειάζονται. Θα πρέπει να είμαστε αρκετά σίγουροι ότι τα πλεονεκτήματα είναι μεγαλύτερα από τα μειονεκτήματα. Ο κώδικας που εκτελείται από πολλά νήματα τα οποία διαχειρίζονται διαμοιραζόμενους πόρους χρειάζεται πολύ προσοχή. Η αλληλεπίδραση των νημάτων μπορεί να γίνει ιδιαίτερα πολύπλοκη. Τα λάθη που προκύπτουν από τον εσφαλμένο συγχρονισμό νημάτων είναι πολύ δύσκολο να ανιχνευθούν και να διορθωθούν.

Επίσης, όταν η CPU διακόπτει την εκτέλεση ενός νήματος για να εκτελέσει ένα άλλο, πρέπει να αποθηκεύσει κάποια στοιχεία για το τρέχον νήμα (τοπικά δεδομένα, δείκτης εκτέλεσης του προγράμματος κλπ.) και να φορτώσει τα στοιχεία του επόμενου νήματος που θα εκτελέσει. Αυτό ονομάζεται εναλλαγή περιβάλλοντος (context switch) καθώς η CPU μεταβαίνει από το ένα περιβάλλον στο άλλο και έχει επιπτώσεις στην απόδοση. Θα πρέπει να φροντίσουμε να γίνεται μόνο όταν χρειάζεται.

Ένα νήμα επίσης χρειάζεται και κάποιο χώρο στη μνήμη για τα δεδομένα του, καθώς και πόρους του λειτουργικού συστήματος για την διαχείρισή του. Γι' αυτό θα πρέπει πάντα να σκεφτόμαστε τα θετικά και αρνητικά πριν τη δημιουργία νέων νημάτων.

3. Δημιουργία και Διαχείριση Νημάτων

Η πολυνηματική (multithreaded) εκτέλεση είναι ένα βασικό στοιχείο της Java. Κάθε εφαρμογή διαθέτει τουλάχιστον ένα νήμα (περισσότερα νήματα αν συμπεριλάβουμε τα νήματα του συστήματος τα οποία κάνουν διαχείριση μνήμης ή διαχείριση σημάτων - signal handling). Κάθε εφαρμογή ξεκινάει με την εκτέλεση ενός μόνο νήματος το οποίο ονομάζεται κύριο νήμα (*main thread*). Αυτό το νήμα έχει την δυνατότητα δημιουργίας και άλλων νημάτων.

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ • ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Μία εφαρμογή η οποία δημιουργεί ένα νήμα (στιγμιότυπο της κλάσης `Thread`) πρέπει να παρέχει και τον κώδικα που θα εκτελεστεί σε αυτό.

Τα νήματα της Java είναι αντικείμενα όπως και τα υπόλοιπα αντικείμενα της Java με την επιπλέον δυνατότητα εκτέλεσης κώδικα. Η δημιουργία ενός νήματος γίνεται με την χρήση του δομητή της `Thread` ως εξής:

```
Thread thread = new Thread();
```

Για να αρχίσει η εκτέλεσή του νήματος καλούμε την μέθοδο start() για αυτό το αντικείμενο:

```
thread.start();
```

Υπάρχουν δύο τρόποι για να δημιουργήσουμε (δηλ. να ορίσουμε τον κώδικα που θα εκτελέσει) ένα νήμα. Ο ένας είναι να περάσουμε ως παράμετρο στον δομητή (constructor) της Thread μία αναφορά ενός αντικειμένου που υλοποιεί τη διασύνδεση Runnable και ο άλλος τρόπος είναι να κληρονομήσουμε τη κλάση Thread και να δημιουργήσουμε αντικείμενο της υποκλάσης. Και στις δύο περιπτώσεις πρέπει να υπερβούμε (override) τη μέθοδο `run`. Ακολουθούν οι δύο προσεγγίσεις:

- Υλοποίηση της διασύνδεσης (interface) `Runnable`.

Η διασύνδεση `Runnable` ορίζει μόνο μία μέθοδο, τη μέθοδο `run`. Η κλάση που θα υλοποιεί την διασύνδεση θα πρέπει να υπερβεί (override) την `run` έτσι ώστε να περιέχει τον κώδικα που θέλουμε να εκτελέσει το νήμα που δημιουργείται. Ένα αντικείμενο τύπου Runnable δίνεται ως αναφορά στον δομητή (constructor) της `Thread`, όπως φαίνεται στο παρακάτω παράδειγμα:

```
public class MyHelloRunnable implements Runnable {
    public void run() {
        System.out.println("Hello Runnable thread!");
    }
    public static void main(String args[]) {
        Thread mHR = new Thread(new MyHelloRunnable());
        mHR.start();
        /* or
        (new Thread(new MyHelloRunnable())).start();
        */
```

```
    }
}
```

Η κλάση MyHelloRunnable υλοποιεί τη διασύνδεση Runnable. Δημιουργούμε το αντικείμενο-νήμα mHR δημιουργώντας στον δομητή της Thread ένα αντικείμενο της κλάσης MyHelloRunnable και μέσω αυτού καλούμε τη μέθοδο start(). Εναλλακτικά μπορούμε να δημιουργήσουμε άμεσα ένα νήμα, χωρίς αναφορά καλώντας την start() με άμεση δημιουργία αντικειμένου Thread περνώντας στο δομητή (constructor) της ένα αντικείμενο τύπου MyHelloRunnable (ο κώδικας που είναι σε σχόλιο). Η μέθοδος run εμφανίζει το μήνυμα "Hello Runnable thread!".

- Να κληρονομήσει/επεκτείνει την κλάση Thread.

Η κλάση Thread υλοποιεί τη διασύνδεση Runnable, επομένως διαθέτει τη μέθοδο run η οποία όμως δεν κάνει κάτι. Έτσι, μία κλάση που θα κληρονομήσει τη κλάση Thread θα πρέπει να παρέχει τη δική της υλοποίηση της run, όπως στο παρακάτω παράδειγμα:

```
public class MyHelloThread extends Thread {
    public void run() {
        System.out.println("Hello my thread!");
    }
    public static void main(String args[]) {
        MyHelloThread mHT = new MyHelloThread();
        mHT.start();
        /* or
        (new MyHelloThread()).start();
        */
    }
}
```

Η κλάση MyHelloThread κληρονομεί την Thread και υπερβαίνει τη μέθοδο run. Δημιουργούμε το αντικείμενο-νήμα mHT και μέσω αυτού καλούμε τη

μέθοδο `start()`. Εναλλακτικά μπορούμε να δημιουργήσουμε άμεσα ένα νήμα, χωρίς αναφορά καλώντας την `start()` με άμεση δημιουργία αντικειμένου `MyHelloThread` (ο κώδικας που είναι σε σχόλιο). Η μέθοδος `run` εμφανίζει το μήνυμα `"Hello my thread!!"`.

Σημειώστε ότι και στα δύο παραδείγματα καλείται η μέθοδος `start()` για να αρχίσει η εκτέλεση του νέου νήματος. Η μέθοδος `start()` τελειώνει και επιστρέφει μόλις αρχίσει το νήμα. Δεν περιμένει να τελειώσει η μέθοδος `run`. Η μέθοδος `run` εκτελείται σαν να εκτελείται από διαφορετική CPU.

Ποιον από τους δύο τρόπους είναι καλύτερο να χρησιμοποιούμε; Και οι δύο τρόποι λειτουργούν. Ο πρώτος όμως τρόπος που χρησιμοποιεί τη διασύνδεση (interface) `Runnable` είναι πιο γενικός επειδή η νέα κλάση που θα υλοποιεί τη διασύνδεση μπορεί ταυτόχρονα να κληρονομήσει κάποια κλάση. Ο δεύτερος τρόπος ίσως είναι πιο εύκολος αλλά περιορίζεται από το γεγονός ότι η νέα κλάση ήδη κληρονομεί την κλάση `Thread` και έτσι δεν μπορεί να κληρονομήσει και άλλη κλάση. Έτσι θα χρησιμοποιήσουμε κυρίως τον πρώτο τρόπο. Ο τρόπος αυτός χρησιμοποιείται και από το υψηλού επιπέδου API διαχείρισης των νημάτων.

Προσοχή, η δημιουργία και η έναρξη εκτέλεσης ενός νέου νήματος πρέπει πάντα να γίνεται με την μέθοδο `start()`. Ένα συνηθισμένο λάθος είναι η χρήση της μεθόδου `run()` αντί της `start()`. Πχ.

```
Thread newThread = new Thread(MyRunnable());
thread.run();  //should be start();
```

Αρχικά μπορεί να μην παρατηρήσουμε κάτι επειδή η μέθοδος `run()` εκτελείται κανονικά. Όμως δεν εκτελείται στο νέο νήμα που μόλις δημιουργήσαμε, αλλά συνεχίζει να εκτελείται στο παλιό νήμα. Γι' αυτό θυμόμαστε ότι η έναρξη εκτέλεσης του νέου νήματος γίνεται πάντα με την μέθοδο `start()`.

Κάθε νήμα έχει ένα όνομα για να μπορεί να αναγνωριστεί. Το όνομα μπορεί να μην είναι μοναδικό, δηλαδή μπορούν πολλά νήματα να έχουν το ίδιο όνομα. Εάν δεν καθορίσουμε το όνομα του νήματος κατά τη δημιουργία του, τότε δημιουργείται

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ • ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

αυτόματα ένα όνομα για αυτό. Το όνομα που δίνεται αυτόματα είναι της μορφής "Thread-<αριθμός>".

Ακολουθεί ένα απλό παράδειγμα. Αρχικά εμφανίζει το όνομα του νήματος που εκτελεί την main(). Αυτό το νήμα ανατίθεται από την εικονική μηχανή της Java (JVM). Μετά δημιουργεί και ξεκινάει 10 νήματα στα οποία δίνει από ένα αριθμό ως όνομα `("Νήμα-" + i)`, περνώντας το όνομα ως παράμετρο στον κατάλληλο δομητή της Thread. Μετά, κάθε νήμα εμφανίζει το όνομά του και τελειώνει την εκτέλεσή του.

```
public class ThreadExample {
  public static void main(String[] args){
   System.out.println(Thread.currentThread().getName());
   for(int i=0; i<10; i++){
     new Thread("Νήμα-" + i){
       public void run(){
         System.out.println("Εκτελείται το: " + getName()
                                             + " running");
       }
     }.start();
   } // for
  } //main
}
```

Η μέθοδος `run()` ορίζεται κατά την δημιουργία του νήματος και η μέθοδος `start()` καλείται άμεσα για κάθε αντικείμενο-νήμα με την δημιουργία του.

Σημειώνεται ότι αν και τα νήματα αρχίζουν να εκτελούνται με τη σειρά, μπορεί να μην εκτελούνται με την ίδια σειρά που δημιουργήθηκαν και ξεκίνησαν, δηλ. μπορεί να μην εμφανιστούν οι αριθμοί 0-9 με τη σειρά. Αυτό συμβαίνει γιατί τα νήματα εκτελούνται παράλληλα και όχι σειριακά. Η JVM και/ή το λειτουργικό σύστημα αποφασίζουν την σειρά με την οποία θα εκτελεστούν και αυτή η σειρά δεν είναι απαραίτητο να είναι η σειρά με την οποία ξεκίνησαν.

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ • ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Κάθε νήμα έχει ένα αριθμό προτεραιότητας. Τα νήματα με υψηλότερη προτεραιότητα εκτελούνται κατά προτίμηση έναντι των νημάτων με μικρότερη προτεραιότητα. Όταν δημιουργείται ένα νέο νήμα, η προτεραιότητά του τίθεται ίση με την προτεραιότητα του νήματος που το δημιουργεί. Οι προτεραιότητες των νημάτων είναι μεταξύ MIN_PRIORITY (με τιμή 1) και MAX_PRIORITY (με τιμή 0). Εξορισμού σε κάθε νήμα δίνεται η τιμή προτεραιότητας NORM_PRIORITY (με τιμή 5).

Κάθε νήμα μπορεί επίσης να οριστεί ως νήμα-δαίμονας (daemon thread). Το νέο νήμα είναι νήμα-δαίμονας εάν και το νήμα που το δημιουργεί είναι και αυτό νήμα-δαίμονας.

Ένα νήμα-δαίμονας έχει τις παρακάτω ιδιαιτερότητες:

- παρέχει υπηρεσίες υποστήριξης διεργασιών παρασκηνίου στα νήματα του χρήστη και αυτός είναι και ο μόνος λόγος ύπαρξής του.
- η διάρκεια ζωής του εξαρτάται από τα υπόλοιπα νήματα του χρήστη. Η JVM τερματίζει το νήμα δαίμονα αν δεν υπάρχουν άλλα νήματα χρήστη.
- Έχει χαμηλή προτεραιότητα.

3.1. Περιγραφή μεθόδων της κλάσης Thread

Η κλάση `Thread` διαθέτει μεθόδους οι οποίες είναι πολύ χρήσιμες στη διαχείριση των νημάτων. Αυτές περιλαμβάνουν `static` μεθόδους οι οποίες παρέχουν πληροφορίες για την κατάσταση ενός νήματος, ή επηρεάζουν την κατάσταση του νήματος. Οι υπόλοιπες μέθοδοι (αυτές που δεν είναι static) καλούνται για να διαχειριστούν το νήμα και το αντικείμενο `Thread`.

Οι μέθοδοι `static` εκτελούν την λειτουργία τους στο τρέχον νήμα. Ακολουθεί η περιγραφή των σημαντικότερων από αυτές:

- public static Thread currentThread()

Επιστέφει μια αναφορά στο τρέχον νήμα, δηλ. το νήμα που κάλεσε αυτή τη μέθοδο.

- public static boolean holdsLock(Object obj)

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ • ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Επιστέφει true εάν το τρέχον νήμα κατέχει το κλείδωμα (lock) του αντικειμένου obj.

- public static boolean interrupted(): Ελέγχει εάν το τρέχον νήμα έχει διακοπεί.
- public static void sleep(long millisec): Προκαλεί τη διακοπή εκτέλεσης του νήματος τουλάχιστον για τον προκαθορισμένο αριθμό milliseconds που δίνουμε ως παράμετρο.
- public static void yield(): Στέλνει σήμα στον χρονοδρομολογητή νημάτων να σταματήσει την εκτέλεση του τρέχοντος νήματος και να αρχίσει την εκτέλεση άλλου νήματος.

Ακολουθούν οι σημαντικότερες μέθοδοι της κλάσης Thread οι οποίες διαχειρίζονται ένα νήμα ή ένα αντικείμενο Thread.

- public String getName(): Επιστρέφει το όνομα του νήματος
- public int getPriority(): Επιστρέφει την προτεραιότητα του νήματος
- public void interrupt(): Διακόπτει την εκτέλεση του νήματος.
- public boolean isAlive(): Ελέγχει εάν το νήμα είναι ζωντανό.
- public boolean isDaemon(): Ελέγχει εάν το νήμα είναι νήμα-δαίμονας.
- public boolean isInterupted(): Ελέγχει εάν έχει διακοπεί η εκτέλεση του νήματος.
- public final void join(long millisec): Το τρέχον νήμα καλεί τη μέθοδο για ένα δεύτερο νήμα, προκαλώντας την εμπόδιση εκτέλεσης (block) του τρέχοντος νήματος μέχρι να τερματιστεί το δεύτερο ή να περάσει ο καθορισμένος αριθμός των milliseconds.
- public void run(): Περιλαμβάνει τον κώδικα που θα εκτελέσει το νήμα.
- public final void setDaemon(boolean on): Μία παράμετρος με τιμή true δηλώνει το νήμα ως νήμα-δαίμονα.
- public final void setName(String name): Θέτει το όνομα του νήματος Thread.

- public final void setPriority(int priority): Θέτει την προτεραιότητα του νήματος μεταξύ 0 και 1
- public void start(): Εκκίνηση ενός νήματος και κλήση της μεθόδου run().

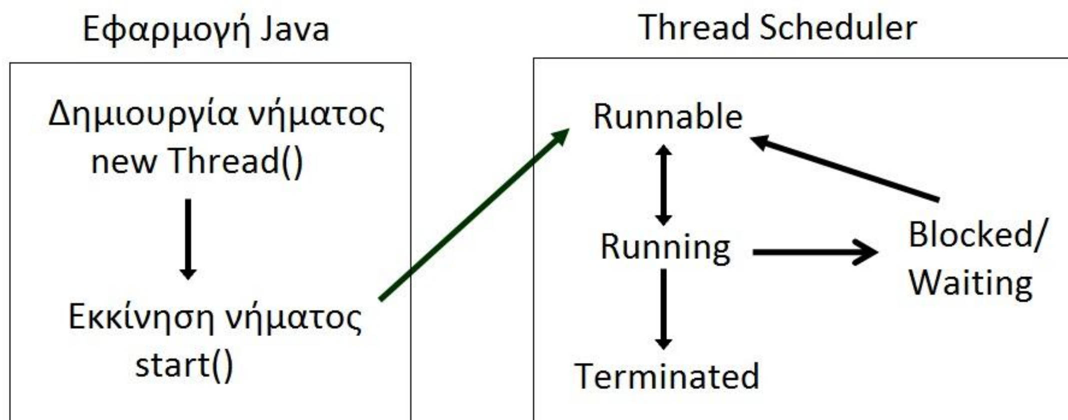
3.2. Χρόνος ζωής ενός νήματος

Είναι σημαντικό να γνωρίζουμε τον κύκλο ζωής ενός νήματος (Life cycle of a Thread) όταν προγραμματίζουμε κάποια πολυνηματική εφαρμογή.

Ένα νήμα μπορεί να βρίσκεται σε μία μόνο από τις παρακάτω καταστάσεις όπως φαίνεται και στο σχήμα 1.

- Δημιουργία (New)

Όταν δημιουργούμε ένα νήμα με τον τελεστή new και πριν την κλήση της start(), το νήμα βρίσκεται σε αυτή την κατάσταση.



Σχ. 1: Κύκλος ζωής νήματος

- Runnable

Όταν καλούμε τη start() η κατάσταση του νήματος αλλάζει σε Runnable και ο έλεγχος μεταφέρεται στο χρονοδρομολογητή των νημάτων (Thread scheduler), για να αποφασίσει για τη συνέχεια. Όταν ένα νήμα μεταφέρεται σε αυτή τη κατάσταση αρχικά το νήμα βρίσκεται στην υποκατάσταση ready, η οποία δηλώνει ότι το νήμα είναι έτοιμο προς εκτέλεση. Ο χρονοδρομολογητής νημάτων (ένα τμήμα της JVM) θα αποφασίσει πότε θα αρχίσει η εκτέλεσή του.

- Running

Το νήμα βρίσκεται σε αυτή την κατάσταση όταν εκτελείται. Ο χρονοδρομολογητής νημάτων επιλέγει ένα από τα νήματα που είναι έτοιμα προς εκτέλεση, αλλάζει την κατάστασή τους σε Running και η CPU αρχίζει την εκτέλεσή του. Από αυτή την κατάσταση, το νήμα μπορεί να αλλάξει την κατάστασή του σε Runnable, Blocked, ή Terminated, ανάλογα με τον χρόνο που του έχει δοθεί, την ολοκλήρωσή του ή τους διαθέσιμους πόρους.

- Blocked/Waiting

Εδώ μιλάμε για δύο καταστάσεις στις οποίες μπορεί να βρίσκεται ένα νήμα. Και στις δύο, το νήμα περιμένει για κάτι πριν αλλάξει την κατάστασή του σε Runnable. Η κατάσταση blocked σχετίζεται με τον συγχρονισμό των νημάτων.

Το νήμα βρίσκεται σε κατάσταση χρονομετρημένης αναμονής (*timed waiting*) όταν περιμένει για συγκεκριμένο χρόνο λόγω κλήση κάποιας μεθόδου χρονομετρημένης αναμονής όπως:

  - Thread.sleep(sleeptime)
  - Object.wait(timeout)
  - Thread.join(timeout)
  - LockSupport.parkNanos(timeout)
  - LockSupport.parkUntil(timeout)

Το νήμα βρίσκεται επίσης σε κατάσταση αναμονής (*waiting*) όταν περιμένει να ελευθερωθεί κάποιος πόρος τον οποίο χρησιμοποιεί κάποιο άλλο νήμα. Όταν το άλλο νήμα τελειώσει ενημερώνει το πρώτο νήμα για να συνεχίσει την εκτέλεσή του πηγαίνοντας στην κατάσταση Runnable. Για παράδειγμα ένα νήμα που έχει καλέσει την Object.wait() για ένα αντικείμενο περιμένει μέχρι ένα άλλο νήμα να καλέσει την Object.notify() ή την Object.notifyAll() για αυτό το αντικείμενο. Ένα νήμα που έχει καλέσει την Thread.join() περιμένει ένα άλλο νήμα να τελειώσει την εκτέλεσή του.

- Terminated

Όταν ένα νήμα ολοκληρώνει την εκτέλεση της run() μεταβαίνει σε αυτή την κατάσταση.

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ • ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΣΠΑ 2007-2013 πρόγραμμα για την ανάπτυξη ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

4. Συγχρονισμός νημάτων

Σε μια πολυνηματική εφαρμογή όπου πολλά νήματα εκτέλεσης διαχειρίζονται τους ίδιους πόρους, θα πρέπει να υπάρχει συγχρονισμός των διεργασιών έτσι ώστε να υπάρχει η σωστή πρόσβαση στους διαθέσιμους πόρους, αλλιώς τα αποτελέσματα ενδέχεται να είναι καταστροφικά. Αν, για παράδειγμα, ένα νήμα διαβάζει μια μεταβλητή ενός αντικειμένου την ίδια στιγμή που ένα άλλο νήμα εκχωρεί μια τιμή σε αυτή, το αποτέλεσμα και των δύο ενεργειών είναι απρόβλεπτο.

Γενικά το πρόβλημα συγχρονισμού διεργασιών είναι αρκετά πολύπλοκο και δύσκολο, αφού εκτός από το να παρέχει τη δυνατότητα αποκλειστικής χρήσης των διαθέσιμων πόρων, ένας μηχανισμός συγχρονισμού πρέπει να εξασφαλίζει και την αποφυγή αδιεξόδων.

Η Java παρέχει έναν σχετικά απλό και εύχρηστο μηχανισμό για το συγχρονισμό των νημάτων. Ο μηχανισμός αυτός βασίζεται στην έννοια του συγχρονισμού και του κλειδώματος (lock). Ο συγχρονισμός στη Java βασίζεται σε μια εσωτερική οντότητα γνωστή ως *κλείδωμα (lock).* Κάθε αντικείμενο μπορεί να έχει ένα δικό του κλείδωμα. Εξ' ορισμού, όταν ένα νήμα χρειάζεται αποκλειστική και συνεπή πρόσβαση στο αντικείμενο τότε *καταλαμβάνει* το αντίστοιχο κλείδωμα, εκτελεί τις λειτουργίες του και όταν τελειώσει *ελευθερώνει* το κλείδωμα. Λέμε ότι το νήμα *κατέχει* το κλείδωμα από τη στιγμή που το καταλαμβάνει μέχρι τη στιγμή που το ελευθερώνει. Όσο το νήμα κατέχει το κλείδωμα, κανένα άλλο νήμα δεν έχει πρόσβαση στο αντικείμενο. Όποιο νήμα προσπαθήσει να καταλάβει το κλείδωμά του, αναστέλλεται και περιμένει την ελευθέρωση του κλειδώματος. Το κλείδωμα επιλύει πιθανές συνθήκες ανταγωνισμού και εξασφαλίζει συνθήκες προήγησης.

Η Java παρέχει δύο επίπεδα συγχρονισμού: *συγχρονισμένες μεθόδους (synchronized methods)* και *συγχρονισμένα τμήματα εντολών (synchronized blocks).*

Η χρήση συγχρονισμένων τμημάτων (synchronized blocks) επιτρέπει τον συντονισμό των διαφόρων νημάτων εκτέλεσης. Στα συγχρονισμένα τμήματα έχουμε

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ • ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

του διαμοιραζόμενους πόρους. Παρακάτω δίνεται η γενική μορφή μιας τέτοιας πρότασης συγχρονισμού (synchronized statement):

```
synchronized(objectidentifier) {
    // Πρόσβαση σε διαμοιραζόμενους πόρους
}
```

Εδώ, το `objectidentifier` είναι μια αναφορά σε ένα αντικείμενο του οποίου το κλείδωμα συνδέεται με τον πόρο που αναπαριστάται στην πρόταση συγχρονισμού.

Οι συγχρονισμένες μέθοδοι παρέχουν αποκλειστική πρόσβαση σε συγκεκριμένο πόρο. Κάθε αντικείμενο της Java θεωρείται ότι διαθέτει ένα κλειδί. Για να υπάρχει αποκλειστική πρόσβαση των μεθόδων ενός αντικειμένου σε κάποιο πόρο, αυτές πρέπει να δηλωθούν ως synchronized (συγχρονισμένες). Στη συνέχεια για να κληθεί μια συγχρονισμένη μέθοδος ενός πόρου αντικειμένου (υπενθυμίζεται ότι όλοι οι πόροι στη Java είναι αντικείμενα), πρέπει να αποκτηθεί το κλειδί του αντικειμένου από το νήμα εκτέλεσης που πραγματοποιεί την κλήση. Το κλειδί επιστρέφεται μετά την εκτέλεση της μεθόδου.

4.1 Συγχρονισμός τμήματος εντολών για πρόσβαση σε μεταβλητή

Ακολουθεί ένα παράδειγμα πρόσβασης σε μια μεταβλητή μετρητή από τρία διαφορετικά νήματα.

Η μέθοδος `printCount()` θα έπρεπε να εμφανίσει με τη σειρά τις τιμές 4,3,2,1. Η κλήση της γίνεται μέσω της `run()` για κάθε νήμα με τη χρήση του αντικειμένου PD της PrintDemo.

```
class PrintDemo {
   public void printCount(){
     try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Μετρητής: "  + i );
         }
     } catch (Exception e) {
```

```
          System.out.println("Thread  interrupted.");
        }
      }
   }
   class ThreadDemo extends Thread {
      private Thread t;
      private String threadName;
      PrintDemo  PD;
      ThreadDemo( String name,  PrintDemo pd){
          threadName = name;
           PD = pd;
      }
      public void run() {
        PD.printCount();
        System.out.println(threadName + " ...terminating.");
      }
      public void start () {
         System.out.println("Starting " +  threadName );
         if (t == null) {
            t = new Thread (this, threadName);
            t.start ();
         }
      }
   }

   public class TestThreadNoSync {
      public static void main(String args[]) {
         PrintDemo PD = new PrintDemo();
         // Δημιουργία 3 νημάτων
         ThreadDemo T1 = new ThreadDemo( "Νήμα 1 ", PD );
```

```
        ThreadDemo T2 = new ThreadDemo( "Νήμα 2 ", PD );
        ThreadDemo T3 = new ThreadDemo( "Νήμα 3 ", PD );
        T1.start();
        T2.start();
        T3.start();

        // το νήμα 2 περιμένει την ολοκλήρωση του 1
        // το νήμα 3 περιμένει την ολοκλήρωση των 1 και 2
        try {
           T1.join();
           T2.join();
           T3.join();
        } catch( Exception e) {
           System.out.println("Interrupted");
        }
    }
}
```

Τα νήματα δεν συγχρονίζονται, και έτσι οι τιμές του μετρητή δεν εμφανίζονται με τη σειρά. Κάθε φορά που τρέχουμε το πρόγραμμα η έξοδος είναι διαφορετική. Ακολουθεί μία από τις εξόδους του προγράμματος:

```
Starting Νήμα 1
Starting Νήμα 2
Starting Νήμα 3
Μετρητής: 4
Μετρητής: 3
Μετρητής: 4
Μετρητής: 2
Μετρητής: 1
Νήμα 1  ...terminating.
Μετρητής: 3
```

```
Μετρητής: 4
Μετρητής: 3
Μετρητής: 2
Μετρητής: 1
Μετρητής: 2
Μετρητής: 1
Νήμα 3  ...terminating.
Νήμα 2  ...terminating.
```

Για να μπορέσουμε να δούμε την εκτέλεση των νημάτων με τη σειρά θα πρέπει να χρησιμοποιήσουμε συγχρονισμό.

Έτσι για να συγχρονίσουμε την πρόσβαση στο αντικείμενο PD, μετατρέπουμε την `run()` χρησιμοποιώντας ένα τμήμα `synchronized()` και έχουμε την εμφάνιση των τιμών του μετρητή με τη σειρά και για τα τρία νήματα.

```
public void run() {
  synchronized(PD) {
     PD.printCount();
  }
  System.out.println(threadName + " ...terminating.");
}
```

Η έξοδος είναι η ίδια κάθε φορά που εκτελείται το πρόγραμμα:

```
Starting Νήμα 1
Starting Νήμα 2
Starting Νήμα 3
Μετρητής: 4
Μετρητής: 3
Μετρητής: 2
Μετρητής: 1
Νήμα 1  ...terminating.
Μετρητής: 4
Μετρητής: 3
```

```
Μετρητής: 2
Μετρητής: 1
Νήμα 2  ...terminating.
Μετρητής: 4
Μετρητής: 3
Μετρητής: 2
Μετρητής: 1
Νήμα 3  ...terminating.
```

4.2 Συγχρονισμένες μέθοδοι

Τόσο οι μέθοδοι static όσο και οι μέθοδοι αντικειμένων μπορεί να είναι συγχρονισμένες (synchronized). Για παράδειγμα η παρακάτω μέθοδος add:

```
public synchronized void add(int value){
    this.count += value;
}
```

Η χρήση της λέξης κλειδί synchronized στη δήλωση της μεθόδου ενημερώνει τη Java ότι η μέθοδος είναι συγχρονισμένη. Μόνο ένα νήμα μπορεί να εκτελείται σε μια συγχρονισμένη μέθοδο κάποιου αντικειμένου. Εάν υπάρχουν περισσότερα αντικείμενα, τότε μόνο ένα νήμα τη φορά μπορεί να εκτελείται σε μια μέθοδο. Ένα νήμα ανά αντικείμενο.

Ακολουθεί ένα παράδειγμα το οποίο αρχίζει 2 νήματα

Here is an example that starts 2 threads and have both of them call the add method on the same instance of Counter. Only one thread at a time will be able to call the add method on the same instance, because the method is synchronized on the instance it belongs to.

```
public class Counter{
   private long count = 0;
   public synchronized void add(long value){
     this.count += value;
   }
```

```
}
public class CounterThread extends Thread{
   private Counter counter = null;
   public CounterThread(Counter counter){
      this.counter = counter;
   }
   public void run() {
      for(int i=0; i<10; i++){
         counter.add(i);
      }
   }
}
public class Example {
  public static void main(String[] args){
    Counter counter = new Counter();
    Thread  t1 = new CounterThread(counter);
    Thread  threadB = new CounterThread(counter);
    t1.start();
    t2.start();
  }
}
```

Δημιουργούνται δύο νήματα. Το ίδιο αντικείμενο counter δίνεται ως παράμετρος στον δομητή (constructor) και των δύο αντικειμένων. Η μέθοδος `Counter.add()` συγχρονίζεται με το αντικείμενο, επειδή η μέθοδος add έχει δηλωθεί ως synchronized() και είναι μέθοδος αντικειμένου. Έτσι μόνο ένα νήμα από τα νήματα μπορεί να καλέσει τη μέθοδο add(). Το άλλο νήμα θα πρέπει να περιμένει μέχρι το πρώτο να ελευθερώσει την μέθοδο add() method, πριν μπορέσει να την εκτελέσει.

Εάν τα δύο νήματα είχαν χρησιμοποιήσει διαφορετικά αντικείμενα `Counter`, θα μπορούσαν να καλέσουν την μέθοδο add() ταυτόχρονα. Οι κλήσεις θα ήταν σε

διαφορετικά αντικείμενα, και έτσι οι μέθοδοι θα συγχρονίζονταν σε διαφορετικά αντικείμενα (το αντικείμενο που κατέχει το κλείδωμα της μεθόδου). Έτσι η μία μέθοδος δεν θα εμπόδιζε την εκτέλεση της άλλης. Ο κώδικας θα μπορούσε να είναι:

```
public class Example {
  public static void main(String[] args){
    Counter counterA = new Counter();
    Counter counterB = new Counter();
    Thread  t1 = new CounterThread(counterA);
    Thread  t2 = new CounterThread(counterB);
    t1.start();
    t2.start();
  }
}
```

Σημειώστε ότι τα δύο νήματα `t1` και `t2`, δεν αναφέρονται στο ίδιο αντικείμενο `Counter`. Η μέθοδος `add` των αντικειμένων `counterA` και `counterB` είναι συγχρονισμένες στα δικά τους αντικείμενα. Η κλήση της `add()` για το `counterA` δεν θα εμποδίσει την κλήση της `add()` για το `counterB`.

5. Ζωτικότητα (Liveness)

Η δυνατότητα μιας πολυνηματικής εφαρμογής ή γενικότερα μιας ταυτόχρονης (concurrent)εφαρμογής να εκτελείται χωρίς υπερβολικές καθυστερήσεις (έγκαιρα) ονομάζεται *ζωτικότητα* της εφαρμογής. Τα πιο κοινά προβλήματα, τα οποία επηρεάζουν τη ζωτικότητα των εφαρμογών είναι το αδιέξοδο (deadlock), το ενεργό αδιέξοδο (livelock) και η παρατεταμένη στέρηση/λιμοκτονία (starvation).

5.1. Αδιέξοδο (Deadlock)

Αδιέξοδο ονομάζεται η κατάσταση που δύο ή περισσότερα νήματα αναστέλλουν την εκτέλεσή τους, καθώς το ένα περιμένει το άλλο. Το ένα νήμα προσπαθεί να καταλάβει ένα κλείδωμα που κατέχει άλλο νήμα, και αντίστροφα.

Αδιέξοδο μπορεί να συμβεί όταν περισσότερα του ενός νήματα χρειάζονται τα ίδια κλειδώματα, την ίδια χρονική στιγμή αλλά τα καταλαμβάνουν με διαφορετική σειρά. Για παράδειγμα, έστω ότι το νήμα thread1 κατέχει το κλείδωμα του πόρου A και προσπαθεί να καταλάβει το κλείδωμα του πόρου B. Ταυτόχρονα, το νήμα thread2 έχει ήδη καταλάβει το κλείδωμα του πόρου B και προσπαθεί να καταλάβει το κλείδωμα του πόρου A. Το νήμα thread1 δεν πρόκειται ποτέ να καταλάβει το B, ενώ το νήμα thread2 δεν πρόκειται ποτέ να καταλάβει το κλείδωμα του A. ΤΟ ένα θα εμποδίζει την εκτέλεση του άλλου. Αυτή η κατάσταση είναι αδιέξοδο.

Ένα χαρακτηριστικό παράδειγμα αδιέξοδου είναι η χρήση των κανόνων ευγενείας από δύο νήματα. Ένας αυστηρός κανόνας ευγένειας απαιτεί ότι όταν υποκλίνεσαι σε ένα φίλο, πρέπει να μείνεις στη στάση υπόκλισης μέχρι να υποκλιθεί και ο φίλος σου. Δυστυχώς αυτός ο κανόνας δεν παίρνει υπόψη του ότι μπορεί και οι δύο φίλοι να υποκλιθούν ταυτόχρονα. Αυτό το παράδειγμα μοντελοποιεί αυτή τη πιθανότητα χρησιμοποιώντας τα δύο νήματα φίλους Alfonse και Gaston.

```
public class Deadlock {
   static class Friend {
      private final String name;
      public Friend(String name) {
         this.name = name;
      }
      public String getName() {
         return this.name;
      }
      public synchronized void bow(Friend bower) {
         System.out.format("%s: %s has bowed to me!%n",
                           this.name, bower.getName());
         bower.bowBack(this);
      }
      public synchronized void bowBack(Friend bower) {
         System.out.format("%s: %s bowed back to me!%n",
```

```
                        this.name, bower.getName());
        }
    }
    public static void main(String[] args) {
        final Friend alphonse = new Friend("Alphonse");
        final Friend gaston = new Friend("Gaston");
        new Thread(new Runnable() {
            public void run() { alphonse.bow(gaston); }
        }).start();
        new Thread(new Runnable() {
            public void run() { gaston.bow(alphonse); }
        }).start();
    }
}
```

Όταν εκτελείται το πρόγραμμα, τα δύο νήματα για να εκτελέσουν τη μέθοδο `bow` καταλαμβάνουν τα κλειδώματα των συγχρονισμένων μεθόδων που αναφέρονται στα (ξεχωριστά) αντικείμενά τους. Κατόπιν, το κάθε νήμα χωρίς να απελευθερώσει το κλειδώμά του, προσπαθεί να καταλάβει το κλείδωμα των συγχρονισμένων μεθόδων του άλλου νήματος για να εκτελέσει τη μέθοδο `bowBack`. Αυτό θα προκαλέσει αδιέξοδο, αφού τα κλειδώματα έχουν ήδη καταληφθεί.

5.2. Ενεργό αδιέξοδο (livelock) και Λιμοκτονία (Starvation)

Τόσο το ενεργό αδιέξοδο(livelock) όσο και η Λιμοκτονία (starvation) συμβαίνουν πολύ πιο σπάνια από ότι το αδιέξοδο, αν και αποτελούν προβλήματα που μπορεί να αντιμετωπίσει κάθε προγραμματιστή πολυνηματικών εφαρμογών.

- **Ενεργό Αδιέξοδο**

Ένα νήμα συχνά δρα σε απόκριση ενός άλλου νήματος. Εάν και η δράση του άλλου νήματος είναι απόκριση στη δράση ενός άλλου νήματος τότε μπορεί να έχουμε ενεργό αδιέξοδο (livelock). Όπως και με το αδιέξοδο, έτσι και εδώ τα νήματα δεν μπορούν να προχωρήσουν. Εδώ όμως δεν μπλοκάρονται, απλά είναι πολύ

απασχολημένα απαντώντας το ένα στο άλλο. Ενεργό *αδιέξοδο* ονομάζεται ένα αδιέξοδο όπου τα εμπλεκόμενα νήματα εκτελούν συνεχώς το ίδιο τμήμα κώδικα και καταλήγουν διαρκώς στην ίδια κατάσταση αναστολής. Δηλαδή παρουσιάζουν μια μικρή δραστηριότητα, δεν είναι εντελώς σταματημένα, όμως αυτή η δραστηριότητα δεν μπορεί να τα βγάλει από τη τροχιά του αδιεξόδου.

- Λιμοκτονία (Starvation)

Η λιμοκτονία περιγράφει μια κατάσταση όπου ένα νήμα δεν μπορεί να έχει πρόσβαση σε διαμοιραζόμενους πόρους και δεν μπορεί να συνεχίσει μα αποτέλεσμα την πρακτική αναστολή εκτέλεσής του. Αυτό συμβαίνει όταν διαμοιραζόμενοι πόροι δεν είναι διαθέσιμοι για μεγάλο χρονικό διάστημα επειδή χρησιμοποιούνται από άλλα "άπληστα" νήματα. (starvation). Η πολυνηματική εφαρμογή σε αυτή την περίπτωση χειρίζεται σωστά τα κλειδώματα, αλλά για λόγους προτεραιοτήτων ή άλλων συνθηκών εκτέλεσης της εφαρμογής, ορισμένα νήματα καταλαμβάνουν τα συγκεκριμένα κλειδώματα με πολύ μεγαλύτερη συχνότητα από κάποια άλλα τα οποία "λιμοκτονούν". Για παράδειγμα, υποθέστε ότι ένα αντικείμενο παρέχει μια συγχρονισμένη μέθοδο η οποία συχνά χρειάζεται πολύ χρόνο για να τελειώσει. Εάν ένα νήμα καλεί αυτή τη μέθοδο συχνά, τα άλλα νήματα τα οποία επίσης χρειάζονται επίσης συχνή συγχρονισμένη πρόσβαση στο ίδιο αντικείμενο, θα μπλοκαριστούν.

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΕΘΝΙΚΗΣ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΥΡΩΠΑΪΚΗ ΕΝΩΣΗ • ΣΥΓΧΡΗΜΑΤΟΔΟΤΗΣΗ ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ