ΝΗΜΑΤΑ ΣΤΗ JAVA (1)

Ορισμός

Νήμα (thread) είναι μια ακολουθιακή ροή ελέγχου (δηλ. κάτι που έχει αρχή, ακολουθία εντολών και τέλος) σ' ένα πρόγραμμα.

Αιτία

Η δυνατότητα απομόνωσης (ή αυτονόμησης) κάποιων εργασιών, ώστε να μπορούν να εκτελούνται ταυτόχρονα.

Παρατηρήσεις

- Ένα νήμα δεν είναι πρόγραμμα αφ' εαυτού του.
- Ένα νήμα εκτελείται μέσα σ' ένα πρόγραμμα.
- Ένα μόνο νήμα δεν προσφέρει κάτι το καινούργιο.
- Δύο ή περισσότερα νήματα μπορούν να εκτελούνται ταυτόχρονα πραγματοποιώντας διαφορετικές εργασίες.

ΝΗΜΑΤΑ ΣΤΗ JAVA (2)

Κάθε νήμα αντιπροσωπεύει μια διεργασία. Διεργασίες που εκτελούνται ταυτόχρονα λέγονται ταυτόχρονες διεργασίες (concurrent processes)

Τη δυνατότητα ταυτόχρονης εκτέλεσης διεργασιών την παρέχει το λειτουργικό σύστημα, όπως π.χ. τα MS Windows, Linux. Τα συστήματα αυτά επιτρέπουν την ταυτόχρονη εκτέλεση πολλών προγραμμάτων (multitasking). Π.χ. ενώ διορθώνουμε ένα κείμενο στον επεξεργαστή κειμένου, γίνεται μια εκτύπωση.

Παρόμοια, ένα μόνο πρόγραμμα μπορεί να εκτελεί ταυτόχρονα πολλές διεργασίες (νήματα). Τέτοια προγράμματα αποτελούν πολυνηματικές εφαρμογές (multithreaded applications). Π.χ. ο φυλλομετρητής HotJava είναι μια τέτοια εφαρμογή.

ΝΗΜΑΤΑ ΣΤΗ JAVA (3)

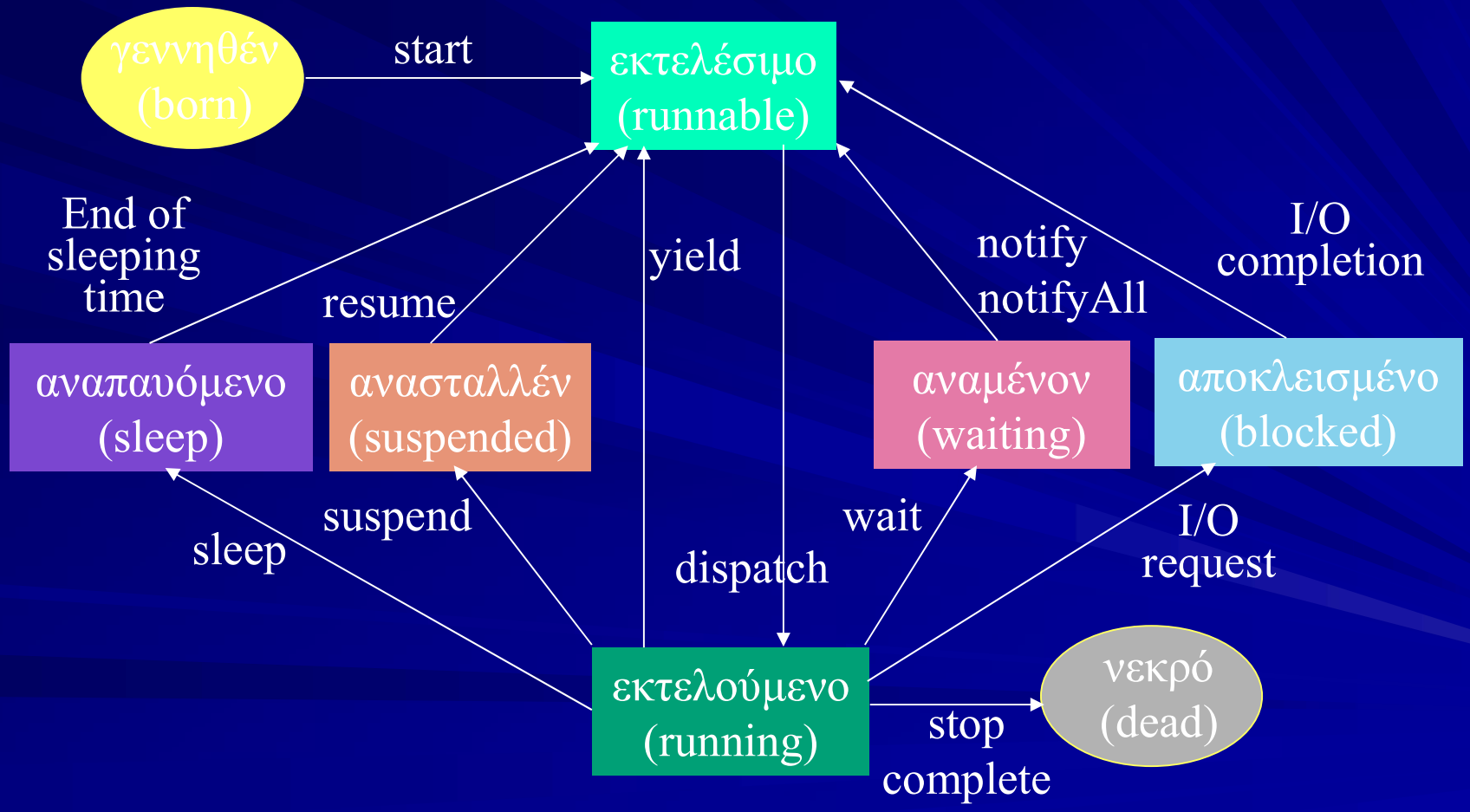
Παραδείγματα ταυτόχρονων διεργασιών:

- Κύλιση μιας σελίδας-κατέβασμα μιας εικόνας
- Εκτέλεση μιας προσομοίωσης-παίξιμο ενός ήχου
- Εκτύπωση μιας σελίδας-κατέβασμα μιας νέας σελίδας

Για να δημιουργήσουμε μια εφαρμογή που χρησιμοποιεί αυτή τη δυνατότητα πρέπει να χρησιμοποιήσουμε μια γλώσσα που να μπορεί να διαχειριστεί ταυτόχρονες διεργασίες (δηλ. νήματα). Μια τέτοια γλώσσα είναι η Java.

Το είδος προγραμματισμού που ασχολείται με τη διαχείριση ταυτόχρονων διεργασιών ονομάζεται ταυτόχρονος προγραμματισμός (concurrent programming).

ΚΥΚΛΟΣ ΖΩΗΣ ΝΗΜΑΤΟΣ



ΔΗΜΙΟΥΡΓΙΑ ΝΗΜΑΤΩΝ ΣΤΗ JAVA

Δύο τρόποι δημιουργίας νημάτων:

(α) Σαν στιγμιότυπα υποκλάσεων της κλάσης Thread

(β) Σαν στιγμιότυπα υλοποιήσεων της διεπαφής Runnable
(όταν η κλάση που δημιουργείται πρέπει να είναι υποκλάση κάποιας άλλης κλάσης, π.χ. της Applet)

Και στις δύο περιπτώσεις πρέπει να οριστεί το σώμα της μεθόδου void run(), που είναι κενό και προσδιορίζει τι κάνει το νήμα.

Η μέθοδος run() είναι (αφηρημένη) μέθοδος της Runnable, αλλά και της Thread (διότι η Thread υλοποιεί την Runnable).

ΚΛΑΣΗ THREAD-ΜΕΘΟΔΟΙ

Άλλες μέθοδοι της Thread είναι:

start() : Απαιτείται η κλήση της για να ενεργοποιηθεί το νήμα.

sleep(n) : Καθορίζει το χρόνο n (σε msec) που (θέλουμε να) καθυστερεί το νήμα πριν αρχίσει η εκτέλεσή του (το n είναι τύπου long). Εγείρει εξαίρεση τύπου InterruptedException.

interrupted() : Επιστρέφει true, αν το νήμα είναι σε διακοπή, αλλιώς false.

interrupt() : Διακόπτει την εκτέλεση του νήματος.

suspend() : Αναστέλλει την εκτέλεση του νήματος.

resume() : Επαναρχίζει η εκτέλεση του νήματος.

stop() : Τερματίζει την εκτέλεση του νήματος.

isAlive() : Επιστρέφει true, αν έχει ενεργοποιηθεί το νήμα (έχει κληθεί η start και δεν έχει τερματιστεί), αλλιώς false.

ΚΛΑΣΗ THREAD-ΔΗΜΙΟΥΡΓΟΙ

Η Thread διαθέτει αρκετούς δημιουργούς. Οι πιο δημοφιλείς είναι:

Thread ()

Thread (String name)

Thread (Runnable object)

Thread (Runnable object, String name)

όπου name είναι το όνομα του νήματος και object ένα αντικείμενο τύπου Runnable.

ΝΗΜΑΤΑ ΜΕ ΧΡΗΣΗ ΤΗΣ THREAD (1)

Διαδικασία δημιουργίας νήματος

- Δημιουργία μιας υποκλάσης της Thread
- Υλοποίηση της run στο σώμα της υποκλάσης.
- Δημιουργία στιγμιοτύπου της υποκλάσης
- Κλήση της start από το στιγμιότυπο

Λειτουργία μεθόδου start

- Δημιουργεί το νέο νήμα εκτέλεσης
- Καλεί τη μέθοδο run
- Επιστρέφει τον έλεγχο στο σημείο κλήσης της

ΝΗΜΑΤΑ ΜΕ ΧΡΗΣΗ ΤΗΣ THREAD (2)

(δημιουργός)

(υλοποίηση της run)

```
public class SimpleThread extends Thread {
    public SimpleThread(String str) {
        super(str);
    }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {
                Thread.sleep((int) (Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName()); }
}
```

ΝΗΜΑΤΑ ΜΕ ΧΡΗΣΗ ΤΗΣ THREAD (3)

```
public class TwoThreadsDemo {
        public static void main (String[] args) {
                thread1 = new SimpleThread("Patra");
                thread2 = new SimpleThread("Athina");
                thread1. start();
                thread2. start();
        }
}
```

(δημιουργία νημάτων ως στιγμιότυπα της SimpleThread και κλήση της start)

ΝΗΜΑΤΑ ΜΕ ΧΡΗΣΗ ΤΗΣ THREAD (4)

```
public class TwoThreadsDemo {
        public static void main (String[] args) {
                new SimpleThread("Patra").start();
                new SimpleThread("Athina").start();
        }
}
```

(δημιουργία νημάτων ως στιγμιότυπα της SimpleThread και κλήση της start)

ΕΚΤΕΛΕΣΗ ΝΗΜΑΤΩΝ

0 Patra
0 Athina
1 Athina
1 Patra
2 Patra
2 Athina
3 Athina
3 Patra
4 Patra
4 Athina
5 Patra
5 Athina
6 Athina
6 Patra
7 Patra
7 Athina
8 Athina
8 Patra
9 Athina
9 Patra
DONE! Athina
DONE! Patra

ΝΗΜΑΤΑ ΜΕΣΩ ΤΗΣ RUNNABLE

Διαδικασία δημιουργίας νήματος

- Δημιουργία κλάσης που υλοποιεί την Runnable
- Υλοποίηση της run στο σώμα της κλάσης.
- Δημιουργία στιγμιοτύπου της κλάσης
- Δημιουργία στιγμιοτύπου της Thread με όρισμα το στιγμιότυπο της κλάσης
- Κλήση της start από το στιγμιότυπο της Thread

ΝΗΜΑΤΑ ΜΕΣΩ ΤΗΣ RUNNABLE (1)

1. Δημιουργία κλάσης – 2. Υλοποίηση run

```
public class CountUp implements Runnable {
    public void run() {
      for (int i=0; i < 10; i++)
         System.out.println("CountUp-" + i);
      }}
```

3. Δημιουργία στιγμιοτύπου της CountUp

```
CountUp counter1 = new CountUp();
```

4. Δημιουργία στιγμιοτύπου Thread με όρισμα δημιουργού counter1

```
Thread thread1 = new Thread(counter1);
```

5. Κλήση μεθόδου start

```
thread1.start();
```

ΝΗΜΑΤΑ ΜΕΣΩ ΤΗΣ RUNNABLE (2)

```
public class Clock extends Applet implements Runnable {
    private Thread clockThread = null;
    public void start() {
        if (clockThread == null) {
            clockThread = new Thread(this, "Clock");
            clockThread.start();
        }
    }
    public void run() {
        Thread myThread = Thread.currentThread();
        while (clockThread == myThread) {
            repaint();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e){}
    }
}
```

ΣΥΓΧΡΟΝΙΣΜΟΣ ΝΗΜΑΤΩΝ (1)

Μέχρι τώρα αναφερθήκαμε σε νήματα εκτελούνται ανεξάρτητα το ένα από το άλλο. Δεν απαιτείται το ένα να γνωρίζει σχετικά με τις εργασίες των άλλων.

Υπάρχουν όμως περιπτώσεις που τα νήματα πρέπει να μοιράζονται δεδομένα. Στην περίπτωση αυτή πρέπει να εξασφαλίσουμε ότι το ένα νήμα δεν θα αλλάξει τα δεδομένα ενόσω τα διαχειρίζεται άλλο νήμα.

Περίπτωση που μπορεί να δημιουργηθεί τέτοιο πρόβλημα υπάρχει όταν διαφορετικά αντικείμενα μιας κλάσης διαχειρίζονται στατικές μεταβλητές ή στατικές μεθόδους της κλάσης.

ΣΥΓΧΡΟΝΙΣΜΟΣ ΝΗΜΑΤΩΝ (2)

Για την αντιμετώπιση τέτοιων καταστάσεων η Java επιτρέπει το κλείδωμα αντικειμένων και μεθόδων με τη χρήση της δεσμευμένης λέξης synchronized.

Έτσι επιτυγχάνεται ο λεγόμενος συγχρονισμός νημάτων, ο οποίος επιτρέπει την προσπέλαση μιας μεθόδου (ή ενός αντικειμένου) σ' ένα μόνο νήμα κάθε φορά. Μια μέθοδος συγχρονίζεται ως εξής:

```
public synchronized void xmethod {…}
```

Για να προσπελάσει ένα νήμα μια μέθοδο που έχει δηλωθεί ως synchronized, πρέπει να περιμένει να τελειώσει το τρέχον νήμα (blocked). (Έλεγχος μέσω monitor object).

ΣΥΓΧΡΟΝΙΣΜΟΣ ΝΗΜΑΤΩΝ (3)

Μπορούμε επίσης να δηλώσουμε ως συγχρονισμένο ένα τμήμα κώδικα στο σώμα μιας μεθόδου, αντί όλης της μεθόδου:

```
public int ymethod {
        …
   synchronized (this) {
        …                }
}
```

ΣΥΓΧΡΟΝΙΣΜΟΣ ΝΗΜΑΤΩΝ (4)

Για μεγαλύτερη ασφάλεια και αποφυγή αδιεξόδων, τα μέρη του κώδικα που έχουν δηλωθεί ως synchronized πρέπει να περιλαμβάνουν κλήσεις της μεθόδου wait():

```
try {
   <κώδικας>
    wait();
}
catch (InterruptedException e) {
}
```

Η κλήση της wait προκαλεί αναστολή της εκτέλεσης του νήματος, μέχρις ότου ένα άλλο νήμα καλέσει τη μέθοδο notify() ή notifyAll() (μεταθέτουν ένα ή όλα τα νήματα από κατάσταση αναστολής σε κατάσταση ετοιμότητας).

ΠΡΟΤΕΡΑΙΟΤΗΤΕΣ ΝΗΜΑΤΩΝ

Υπάρχουν περιπτώσεις που θέλουμε τα νήματα να μην είναι ισοδύναμα, αλλά κάποια να εκτελούνται περισσότερο χρόνο από άλλα.

Αυτό επιτυγχάνεται με τον καθορισμό προτεραιότητας για κάθε νήμα (1-10). Η κανονική προτεραιότητα είναι 5.

Η Thread διαθέτει τρεις σταθερές MAX_PRIORITY, MIN_PRIORITY και NORM_PRIORITY με τιμές 10, 1 και 5.

Η προτεραιότητα ενός νήματος τίθεται με την setPriority (int priority), ενώ με την getPriority() μπορούμε να ανιχνεύσουμε την τρέχουσα τιμή προτεραιότητας ενός νήματος.