Προγραμματισμός σε Java

Νήματα - Συγχρονισμός

Παναγιώτης Αδαμίδης
Τμήμα Μηχανικών Πληροφορικής
Αλεξάνδρειο ΤΕΙ Θεσσαλονίκης
adamidis@it.teithe.gr

Περιεχόμενα

- 1. Διεργασίες και Νήματα (Processes and Threads)
- 2. Πολυνηματικός Προγραμματισμός (Mulithreaded Programming)
- 3. Δημιουργία και Διαχείριση Νημάτων
- 4. Συγχρονισμός νημάτων
- 5. Ζωτικότητα (Liveness)

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

Διεργασίες (Processes) και Νήματα (Threads)

- Βασικά στοιχεία εκτέλεσης στον ταυτόχρονο προγραμματισμό.
- Java: Ο ταυτόχρονος προγραμματισμός έχει να κάνει κυρίως με νήματα (threads).
- Ένα υπολογιστικό σύστημα έχει πολλές ενεργές διεργασίες και νήματα, ακόμη και αν διαθέτει μόνο μία μονάδα επεξεργασίας (time sharing ή time slicing).
- Πολλοί επεξεργαστές (multiple processors) ή επεξεργαστές με πολλούς πυρήνες επεξεργασίας (multiple execution cores).
  - Δυνατότητα εκτέλεσης ταυτόχρονων διεργασιών και νημάτων.

Διεργασίες (Processes)

- Δικό της περιβάλλον εκτέλεσης: . Γενικά, μία διεργασία διαθέτει ένα πλήρες, ιδιωτικό σύνολο βασικών πόρων εκτέλεσης και κυρίως το δικό της χώρο μνήμης.
- Συχνά συνώνυμο των προγραμμάτων ή των εφαρμογών. Όμως αυτό που βλέπουν οι χρήστες ως μία εφαρμογή συνήθως είναι ένα σύνολο συνεργαζόμενων διεργασιών.
- Υποστήριξη επικοινωνίας μεταξύ διεργασιών (*Inter Process Communication* - IPC) από ΛΣ
  - όπως διοχετεύσεις (pipes) και υποδοχές (sockets).
  - επικοινωνία μεταξύ των διαδικασιών του ίδιου συστήματος αλλά και μεταξύ διαφορετικών συστημάτων
- JVM: λειτουργεί ως μία διεργασία.

Νήματα (Threads)

- Όπως οι διεργασίες (processes), έτσι και τα νήματα παρέχουν ένα περιβάλλον εκτέλεσης.
- 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 (πχ. μνήμη και αρχεία). Αυτό αυξάνει την απόδοση και εξασφαλίζει αποτελεσματική, επικοινωνία μεταξύ των νημάτων.

Πολυνηματικός Προγραμματισμός (Mulithreaded Programming)

- Ταυτόχρονη εκτέλεση πολλών νημάτων μέσα στο ίδιο πρόγραμμα.
- Μπορούμε να θεωρήσουμε ένα νήμα που εκτελείται ως μία CPU που εκτελεί το πρόγραμμα.
  - Πολλά νήματα που εκτελούν το ίδιο πρόγραμμα → πολλές CPU που εκτελούν το ίδιο πρόγραμμα.
- Ο πολυνηματισμός (multithreading) είναι ένας πολύ καλός τρόπος να βελτιώσουμε την εκτέλεση κάποιων προγραμμάτων.

Πολυνηματικός Προγραμματισμός Πλεονεκτήματα

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

Πολυνηματικός Προγραμματισμός Μειονεκτήματα

- Πρόσβαση σε διαμοιραζόμενους πόρους
- Εναλλαγή περιβάλλοντος (context switch)
- Χρήση επιπλέον πόρων.
- Πολυπλοκότητα αποσφαλμάτωσης
- Έλλειψη συμβατότητας με υπάρχοντα κώδικα

Δημιουργία και Διαχείριση Νημάτων

- Η πολυνηματική (multithreaded) εκτέλεση είναι ένα βασικό στοιχείο της Java.
- Κάθε εφαρμογή διαθέτει τουλάχιστον ένα νήμα και ξεκινάει με την εκτέλεση ενός μόνο νήματος το *κύριο νήμα* (*main thread*). Αυτό το νήμα έχει την δυνατότητα δημιουργίας και άλλων νημάτων.
- Μία εφαρμογή η οποία δημιουργεί ένα νήμα 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.

Δημιουργία Νημάτων - (1)

- Ένα νήμα είναι στιγμιότυπο της κλάσης Thread.
- Δημιουργία νήματος με χρήση του δομητή της Thread:

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

- Εκκίνηση εκτέλεσης για το αντικείμενο thread :

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

- Υπέρβαση της μεθόδου run()

Δημιουργία Νημάτων - (2)

- Δύο τρόποι δημιουργίας νήματος:
  - χρήση αναφοράς αντικειμένου που υλοποιεί τη διασύνδεση Runnable κατά τη δημιουργία του νήματος στον δομητή (constructor) της Thread.
  - Κληρονομώντας/επεκτείνοντας τη κλάση Thread και την ακόλουθη δημιουργία αντικειμένου της υποκλάσης

Με υλοποίηση Runnable

```
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();
        */
    }
}
```

Με επέκταση Thread

```
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();
        */
    }
}
```

Παράδειγμα: Δημιουργία, εμφάνιση ονόματος νημάτων

```
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
}
```

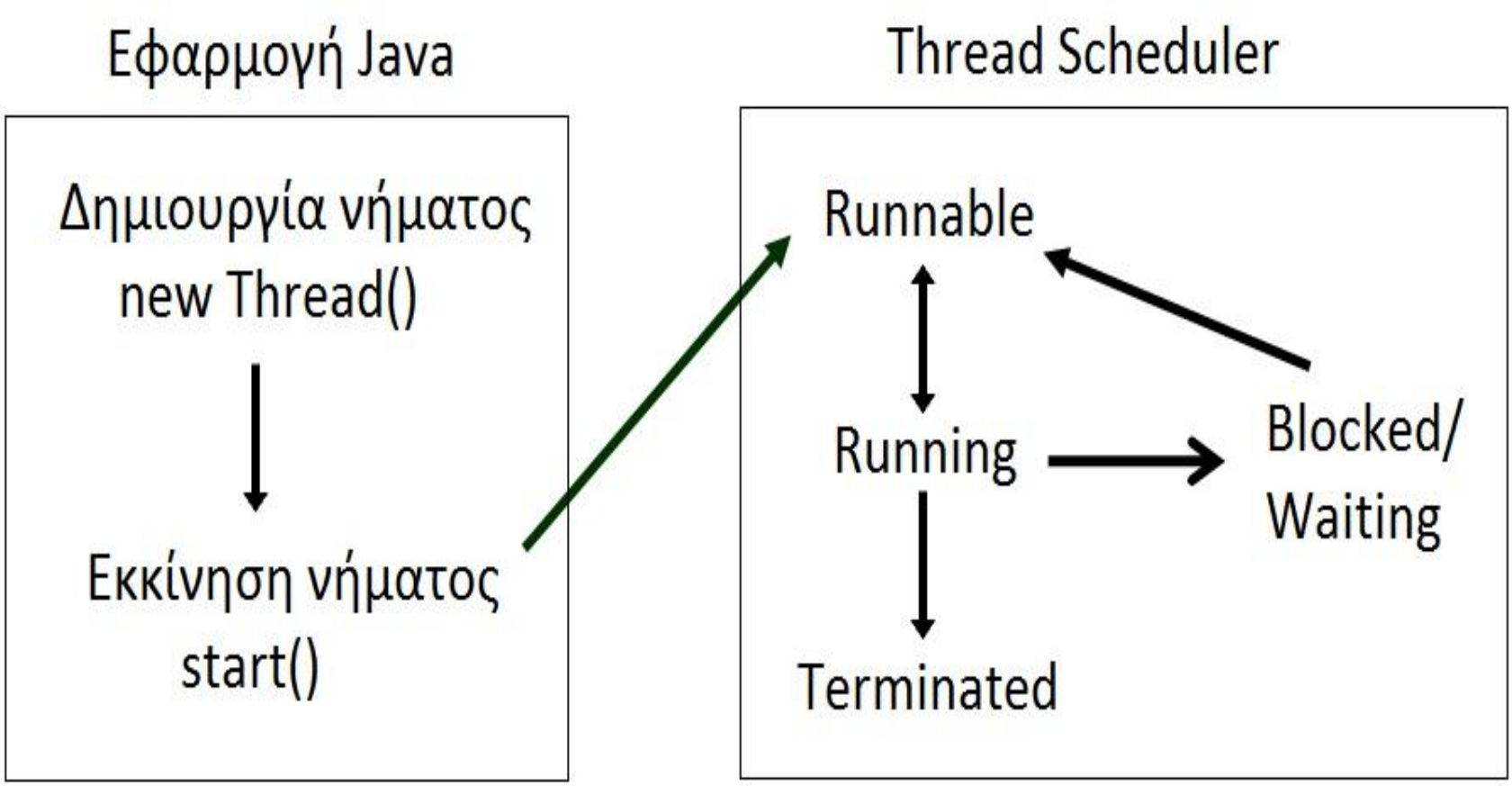
Μέθοδοι κλάσης Thread (static)

- public static Thread currentThread()
- public static boolean holdsLock(Object obj)
- public static boolean interrupted()
- public static void sleep(long millisec)
- public static void yield()

Μέθοδοι κλάσης Thread (instance)

- public String getName()
- public int getPriority()
- public void interrupt()
- public boolean isAlive()
- public boolean isDaemon()
- public boolean isInterupted()
- public final void join(long millisec)
- public void run()
- public final void setDaemon(boolean on)
- public final void setName(String name)
- public final void setPriority(int priority)
- public void start()

Χρόνος ζωής νήματος
Thread life cycle



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.
- 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.

Μηχανισμός συγχρονισμού Java

- Σχετικά απλός και εύχρηστος.
- Βασίζεται στην έννοια του συγχρονισμού και του κλειδώματος (lock).
- Κάθε αντικείμενο έχει ένα δικό του κλείδωμα.
- Όταν ένα νήμα χρειάζεται αποκλειστική πρόσβαση στο αντικείμενο τότε *καταλαμβάνει* το αντίστοιχο κλείδωμα, εκτελεί τις λειτουργίες του και όταν τελειώσει *ελευθερώνει* το κλείδωμα.
- Όσο το νήμα κατέχει το κλείδωμα, κανένα άλλο νήμα δεν έχει πρόσβαση στο αντικείμενο.

Μηχανισμός συγχρονισμού Java (2)

- Το κλείδωμα επιλύει πιθανές συνθήκες ανταγωνισμού και εξασφαλίζει συνθήκες προήγησης.
- Δύο επίπεδα συγχρονισμού: *συγχρονισμένες μεθόδους (synchronized methods)* και *συγχρονισμένα τμήματα εντολών (synchronized blocks)*.

Χρήση συγχρονισμένων τμημάτων κώδικα

- Επιτρέπει τον συντονισμό των διαφόρων νημάτων εκτέλεσης.
- Στα συγχρονισμένα τμήματα έχουμε τους διαμοιραζόμενους πόρους.

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

- Objectidentifier: αναφορά σε ένα αντικείμενο του οποίου το κλείδωμα συνδέεται με τον πόρο που αναπαριστάται στην πρόταση συγχρονισμού.

Παράδειγμα συγχρονισμένου τμήματος κώδικα -1

```
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.");
         }
    }
}
```

Παράδειγμα συγχρονισμένου τμήματος κώδικα - 2

```
class ThreadDemo extends Thread {
    private Thread t;    private String threadName;  PrintDemo  PD;
    ThreadDemo( String name,  PrintDemo pd){
        threadName = name;   PD = pd;
    }
 public void run() {
     synchronized(PD) {
        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 ();
         }
    }
}
```

Παράδειγμα συγχρονισμένου τμήματος κώδικα - 3

```
public class TestThreadNoSync {
    public static void main(String args[]) {
        PrintDemo PD = new PrintDemo();
        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();
         try {
              T1.join();
              T2.join();
              T3.join();
      } catch( Exception e) {  System.out.println("Interrupted"); }
    }
```

Παράδειγμα συγχρονισμένου τμήματος κώδικα - Έξοδος

```
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.
```

Συγχρονισμένες μέθοδοι

- Τόσο οι μέθοδοι static όσο και οι μέθοδοι αντικειμένων μπορεί να είναι συγχρονισμένες (synchronized).

```
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.

Παράδειγμα συγχρονισμένων μεθόδων - 1

```
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);
    }
 }
```

Παράδειγμα συγχρονισμένων μεθόδων - 2

```
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();
   }
 }
```

Παράδειγμα συγχρονισμένων μεθόδων - 3

```
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();
    }
  }
```

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).