Κεφάλαιο 1. Νήματα (Threads).

Time Sharing

Η επεξεργαστική ισχύς είναι ένας πόρος περιορισμένος (ιδιαίτερα στις προηγούμενες δεκαετίες) ο οποίος θέλουμε να εξυπηρετεί ταυτόχρονα πολλές εργασίες. Στους πρώτους υπολογιστές η εξυπηρέτηση πολλών εργασιών γινόταν με batch διαδικασίες όπου ο κάθε ενδιαφερόμενος υπέβαλε την εργασία του (π.χ. ένα πηγαίο πρόγραμμα σε διάτρητες κάρτες με σκοπό να το ελέγξει και να το μεταγλωττίσει) και περίμενε στην ουρά μέχρι να έρθει η σειρά του ώστε, αφού είχαν ολοκληρωθεί οι εργασίες που προηγούταν της δικής του, να εκτελεστεί η δική του. Αυτή ήταν μια πρώτη προσπάθεια για να γίνει διαθέσιμη η υπολογιστική ισχύς σε περισσότερους από έναν χρήστες.

Αργότερα εμφανίστηκαν λειτουργικά συστήματα που υποστήριζαν την ύπαρξη περισσοτέρων του ενός (συνήθως σειριακών) τερματικών (οθόνη και πληκτρολόγιο). Έτσι, ένας (mini ή mainframe υπολογιστής) είχε ταυτόχρονα πολλούς συνδεδεμένους χρήστες που μπορούσαν να υποβάλουν διαφορετικές εργασίες. Ακόμα και αυτό το πρόγραμμα που δεχόταν εντολές από το χρήστη (εντολές που πληκτρολογούσε ο χρήστης) και τις έλεγχε (συντακτικά) αποτελούσε μια εργασία. Έτσι ένα σύστημα (π.χ. ένας mini υπολογιστής) με πέντε τερματικά έπρεπε να εκτελεί ταυτόχρονα πέντε εργασίες (φλοιού ή command prompt) για να μπορεί να εξυπηρετεί ανάλογο πλήθος χρηστών. Σε αυτή τη γενιά συστημάτων εκτός από τις εργασίες για την εξυπηρέτηση των τερματικών μπορεί να έπρεπε να εκτελεστούν και άλλες εργασίες. Το συμπέρασμα είναι ότι υπήρξε ανάγκη να εξυπηρετηθούν ταυτόχρονα πολλές εργασίες. Οι εργασίες αυτές ονομάστηκαν Διεργασίες (Processes) και αποτελούν ακόμα και σήμερα θεμελιώδες παράγοντα σε κάθε λειτουργικό σύστημα. Η ανάγκη για ταυτόχρονη εκτέλεση πολλών διεργασιών αντιμετω-

πίστηκε με το λεγόμενο Time Sharing. Στην πραγματικότητα κάθε διεργασία εκτελούταν για ένα μικρό χρονικό διάστημα και μετά αποσυρόταν για να εκτελεστεί μια άλλη διεργασία. Έτσι όλες οι διεργασίες έμοιαζαν να εκτελούνται ταυτόχρονα. Ενδεικτικά αναφέρουμε ότι όλοι οι συνδεδεμένοι χρήστες είχαν την αίσθηση ότι το σύστημα δουλεύει για αυτούς.

Multi-Tasking

Σε επόμενες γενιές λειτουργικών συστημάτων οι χρήστες απέκτησαν τη δυνατότητα να εκτελούν ταυτόχρονα πολλές εργασίες (προγράμματα). Για παράδειγμα στο λειτουργικό σύστημα Unix μπορούσε να υπάρχει μία εργασία στο foreground (αυτή που αλληλεπιδρούσε με το χρήστη – έστελνε έξοδο στην οθόνη και λάμβανε είσοδο από το πληκτρολόγιο) και πολλές εργασίες στο background (αυτές δεν αλληλοεπιδρούσαν με το χρήστη – έστελναν έξοδο σε κάποιο text αρχείο και λάμβαναν είσοδο από κάποιο άλλο αρχείο). Ορισμένες από τις εντολές στο Unix για τη διαχείριση εργασιών (ακριβέστερα διεργασιών) ήταν οι: <εντολή> &, bg, fg, jobs, kill, ps, κλπ.

Ανάλογες δυνατότητες προσέφεραν και πιο σύγχρονα παραθυρικά λειτουργικά συστήματα. Εδώ ο χρήστης δεν ξεχώριζε εργασίες στο foreground από εργασίες στο background. Τα πάντα ήταν εργασίες που αλληλεπιδρούσαν με το χρήστη χρησιμοποιώντας (συνήθως) διαφορετικά παράθυρα. Τόσο όμως στις εργασίες foreground, background και τα διαφορετικά παράθυρα έχουν πίσω τους (υλοποιούνται με) διεργασίες. Έτσι οι διεργασίες εξελίχθηκαν και εξυπηρετούν πολλές εργασίες για κάθε χρήστη αντί για μία εργασία ανά χρήστη. Όμως και πάλι κοινώς παρονομαστής είναι το Time Sharing. Εδώ εκτελούνται όλες οι διεργασίες (παράθυρα ή εργασίες foreground και background) για ένα μικρό χρονικό διάστημα και μετά αποσύρονται για να εκτελεστεί μια άλλη διεργασία.

Η δυνατότητα να εκτελεί πολλές εργασίες ένας χρήστης (είτε με foreground και background εργασίες, είτε με διαφορετικά παράθυρα) ονομάστηκε Multitasking.

Multi-Threading

Η όρεξη για εξέλιξη δεν σταματά ποτέ. Το επόμενο βήμα σε αυτή την εξέλιξη ονομάστηκε Multithreading. Αυτό βασίζεται στην ιδέα ότι διαφορετικά κομμάτια του ίδιου προγράμματος (για παράδειγμα κάποιες μέθοδοι ενός αντικειμενοστρεφούς προγράμματος) πρέπει να μπορούν να εκτελούνται ταυτόχρονα. Σύμφωνα με αυτή τη λογική μπορούν να υπάρχουν προγράμματα που έχουν σχεδιαστεί έτσι ώστε να αποτελούνται μόνο από ένα Νήμα (Thread) αλλά μπορεί να υπάρχουν και προγράμματα που είναι σχεδιασμένα έτσι ώστε να έχουν διαφορετικά τμήματα

τους που μπορούν να εκτελεστούν ταυτόχρονα και σε ανεξαρτησία το ένα από το άλλο (πολλά Νήματα, πολλά Thread). Για παράδειγμα ένα σχεδιαστικό πρόγραμμα (Computer Aided Design / Computer Aided Manufacturing, CAD/CAM) μπορεί να έχει ένα τμήμα (ένα Thread) που αλληλεπιδρά με ένα χρήστη (π.χ. ένα μηχανολόγο) για να σχεδιάσει το δεύτερο από τα δύο γρανάζια μετάδοσης κίνησης μιας μηχανής και επίσης να έχει ένα άλλο τμήμα (ένα άλλο Thread) που μετασχηματίζει το σχέδιο από το πρώτο (ολοκληρωμένο σχεδιαστικά) γρανάζι κίνησης σε G-εντολές και να στέλνει τις G-εντολές σε ένα CNC μηχανουργικό εργαλείο (CNC τόρνο ή CNC φρέζα) για να παράγει (να κόψει και να μορφοποιήσει) το γρανάζι.

Για πληρότητα αναφέρουμε ότι το CNC είναι ακρώνυμο της φράσης Computer Numerical Control και σημαίνει ότι ένας υπολογιστής μετατρέπει τα σχέδια που έγιναν με τη βοήθεια ενός προγράμματος CAD/CAM σε θεμελιώδης αριθμητικές εντολές η οποίες υποδηλώνουν εντολές μετακίνησης ενός κοπτικού – διαμορφωτικού μηχανουργικού εργαλείου (Lathe – τόρνο, Mill – φρέζα) για να παραχθεί χωρίς ανθρώπινη παρέμβαση (χωρίς μηχανουργό) ένα (συνήθως) μεταλλικό εξάρτημα. Για τον ίδιο λόγο (πληρότητα) αναφέρουμε ότι οι G-εντολές είναι εντολές που μπορεί να καταλάβει μια CNC-μηχανή και (συνήθως) αποτελούνται από ένα λατινικό χαρακτήρα τον οποίο ακολουθεί ένα ακέραιος αριθμός. Επειδή πολύ συχνά χρησιμοποιείται ο χαρακτήρας G (με μικρότερη συχνότητα χρησιμοποιείται ο χαρακτήρας M και με ακόμα μικρότερη άλλοι χαρακτήρες) έχει επικρατήσει η ονομασία G-εντολές και G-code.

Μπορούμε τώρα να αποδώσουμε ετυμολογικά το όρο Multithreading. Ο όρος αυτός αποδίδει το thread-based multitasking. Αναφέραμε παραπάνω ότι τα threads μπορεί να γίνουν κάποιες μέθοδοι ενός αντικειμενοστρεφούς προγράμματος. Στην πραγματικότητα threads μπορούν να γίνουν και άλλα μπλοκ (ακολουθίες εντολών) κώδικα, όπως θα δούμε παρακάτω.

Δημιουργία Νήματος

Στη γλώσσα Java υπάρχουν δύο τρόποι να δημιουργήσουμε Threads. Ο πρώτος βασίζεται στην επέκταση (κληρονομιά) του αντικειμένου Thread και ο δεύτερος βασίζεται στην αξιοποίηση της διασύνδεση (interface) Runnable. Θα εξετάσουμε κάθε μία ξεχωριστά. Στη συνέχεια του κεφαλαίου προκειμένου να διαχωρίσουμε τη γενική έννοια του Νήματος (Thread) από την κλάση Thread που διαθέτει η Java θα κάνουμε μία σύμβαση. Θα χρησιμοποιούμε τη λέξη thread (όλα πεζά) για τη γενική έννοια του Νήματος και θα χρησιμοποιούμε τη λέξη Thread (πρώτο γράμμα κεφαλαίο) για την κλάση που διαθέτει η Java.

Δημιουργία Νήματος επεκτείνοντας την κλάση Thread

Απαιτούνται 3 βασικά βήματα. Δημιουργία της δικιάς μας κλάσης που επεκτείνει (κληρονομεί) την κλάση Thread. Σε αυτή την κλάση τοποθετούμε (κωδικοποιούμε) την δική μας μέθοδο run η οποία πρέπει να περιέχει τον κώδικα που θα εκτελεί το thread. Δεύτερο βήμα είναι η δημιουργία ενός στιγμιότυπου της δικής μας κλάσης απογόνου της Thread. Τρίτο και τελευταίο βήμα είναι η εκκίνηση του thread (νήματος) που αντιστοιχεί στην κλάση μας, με χρήση των μεθόδων (συγκεκριμένα της μεθόδου start) που διαθέτει η κλάση μας (από κληρονομιά εκ της πατρικής κλάσης Thread) για αυτό το σκοπό.

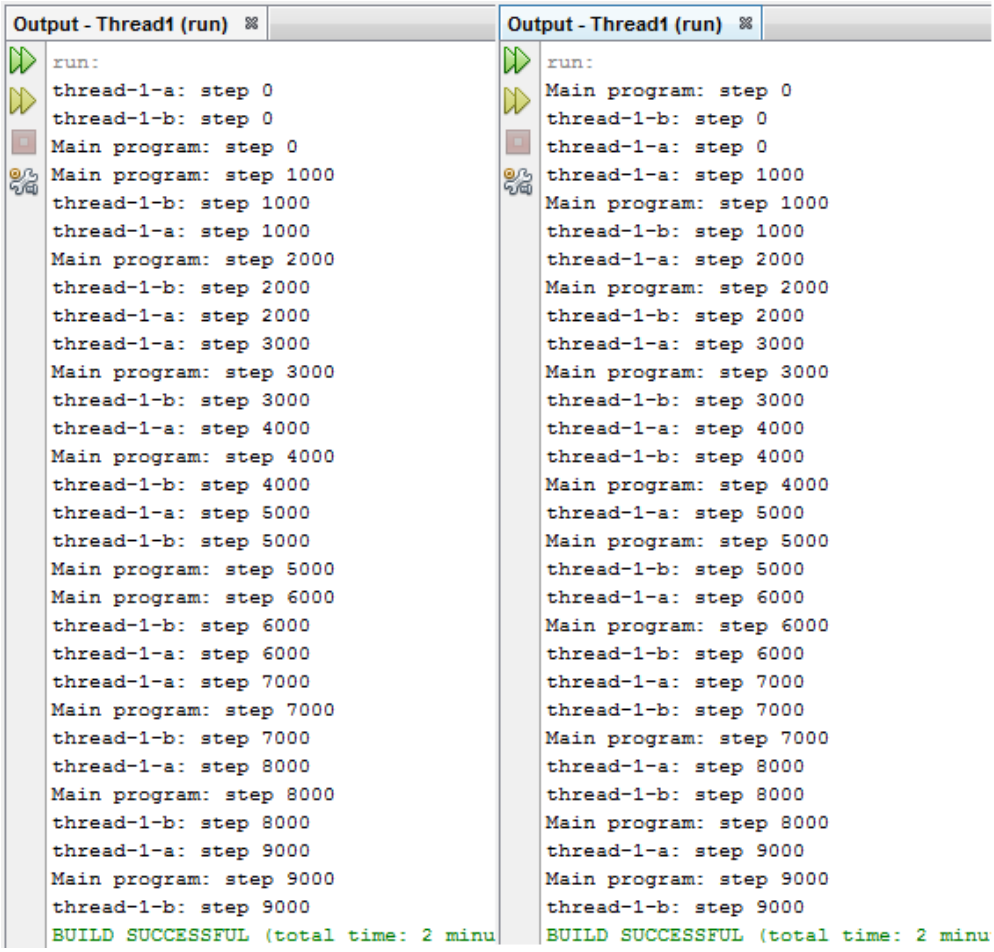
Παράδειγμα:

```
 1. import static java.lang.Thread.sleep;
 2.
 3. public class thread1 extends Thread {
 4.    // constructor
 5.    public thread1 (String name) {
 6.       super(name);
 7.    }
 8.    // κώδικας Thread
 9.    public void run () {
10.       for (int i=0; i<10000; i++) {
11.          try {sleep(10);} catch (Exception e) {};
12.          if (i % 1000 == 0)
13.             System.out.println(this.getName() + ": step "+ i);
14.       }
15.    }
16. }
17.
18. class Main {
19.    public static void main (String argv[]) {
20.       thread1 t1a = new thread1("thread-1-a");
21.       thread1 t1b = new thread1("thread-1-b");
22.       t1a.start();
23.       t1b.start();
24.       for (int i=0; i<10000; i++) {
25.          try {sleep(10);} catch (Exception e) {};
26.          if (i % 1000 == 0)
27.             System.out.println("Main program: step "+ i);
28.       }
29.    }
30. }
```

Στο παράδειγμα αυτό (σε εφαρμογή του πρώτου βασικού βήματος) δημιουργήσαμε τη δική μας κλάση thread1 που επεκτείνει την κλάση Thread (γραμμές 3 – 16). Σε αυτή την κλάση τοποθετήσαμε τη μέθοδο run που περιέχει τον κώδικα που θα εκτελεί το thread (γραμμές 9 – 15). Επιπλέον, συμπληρωματικά στο πρώτο βασικό βήμα, στην κλάση αυτή (thread1) ορίσαμε και τον κατασκευαστή με ένα όρισμα (γραμμές 5 – 7) για να μπορεί το thread όταν δημιουργείται να λαμβάνει το όνομα που εμείς επιθυμούμε καθώς τα system generated ονόματα μπορεί να μην είναι πολύ υποβοηθητικά.

Στη συνέχεια δημιουργήσαμε την κλάση Main (γραμμές 18 – 30, στο ρόλο κυρίου προγράμματος) η οποία διαθέτει τη μέθοδο main και ως εκ τούτου είναι εκείνη που ξεκινάει την εκτέλεση του προγράμματος μας. Αυτή (η μέθοδος main ή αλλιώς το κύριο μας πρόγραμμα) κάνει τρία πράγματα (τρεις ενέργειες). Η πρώτη ενέργεια είναι σύνθετη και υλοποιεί το δεύτερο και τρίτο βασικό βήμα που αναφέρθηκε παραπάνω. Δηλαδή, δημιουργεί το στιγμιότυπο της κλάσης thread1 που ονομάζεται "thread-1-a" (γραμμή 20, πρόκειται για το δεύτερο βασικό βήμα που αναφέρθηκε παραπάνω) και ξεκινάει ένα thread (Νήμα, στη γραμμή 22, πρόκειται για το τρίτο βασικό βήμα που αναφέρθηκε παραπάνω) που θα εκτελέσει τη λειτουργία του "thread-1-a" (αυτή της μεθόδου run). Η δεύτερη ενέργεια της main είναι παρόμοια με την πρώτη. Επειδή έχουμε δύο threads που ξεκινούν από τη main, επαναλαμβάνουμε και δημιουργούμε ένα δεύτερο στιγμιότυπο της κλάσης thread1 που ονομάζεται "thread-1-b" (γραμμή 21) και στη συνέχεια ξεκινάμε ένα thread (Νήμα, στη γραμμή 23) που θα εκτελέσει τη λειτουργία του "thread-1-b". Η τελευταία (τρίτη ενέργεια) που κάνει η main είναι να τρέχει ένα επαναληπτικό σχήμα δέκα χιλιάδων (10000) βημάτων όπου κάθε χίλια βήματα εμφανίζει ένα μήνυμα. Ανάλογη είναι και η εργασία (run) που κάνει και το κάθε thread (thread-1-a και thread-1-b).

Σε αυτό το πρόγραμμα έχουμε 3 thread. Το πρώτο ξεκινάει με την εκτέλεση της main από την JVM. Λίγο μετά (γραμμή 22) ξεκινάει το δεύτερο thread (το thread-1-a) και λίγο πιο μετά (γραμμή 23) ξεκινάει το τρίτο thread (το thread-1-b). Τα thread αυτά ανταγωνίζονται να λάβουν επεξεργαστικούς κύκλους. Έτσι στην εικόνα 1 βλέπουμε την έξοδο από την πρώτη (αριστερά) και δεύτερη (δεξιά) εκτέλεση. Όπως βλέπουμε στην εικόνα 1 παρότι τα 3 thread κάνουν τα ίδια πράγματα και έχουν τους ίδιους χρόνους καθυστέρησης (sleep(10)) δεν έχουν πάντα την ίδια σειρά που εμφανίζουν μηνύματα.



Εικόνα 1. *Έξοδος προγράμματος με 3 threads*

Δημιουργία Νήματος υλοποιώντας το interface Runnable

Απαιτούνται 4 βασικά βήματα. Δημιουργία της δικιάς μας κλάσης που υλοποιεί (implements) τη διασύνδεση Runnable. Σε αυτή την κλάση τοποθετούμε (κωδικοποιούμε) την δική μας μέθοδο run η οποία πρέπει να περιέχει τον κώδικα που θα εκτελεί το thread. Δεύτερο βασικό βήμα είναι η δημιουργία ενός στιγμιότυπου της δικής μας κλάσης. Τρίτο βασικό βήμα είναι η δημιουργία ενός στιγμιότυπου της κλάσης Thread στον κατασκευαστή του οποίου περνάμε ως παράμετρο το στιγμιότυπο της δικιάς μας κλάσης. Το τελευταίο (τέταρτο) βασικό βήμα είναι η εκκίνηση του στιγμιότυπου της κλάσης Thread, με χρήση των μεθόδων (συγκεκριμένα της μεθόδου start) που διαθέτει η κλάση Thread.

Παράδειγμα:

```
 1. import static java.lang.Thread.sleep;
 2.
 3. public class thread2 implements Runnable {
 4.    // κώδικας Thread
 5.    public void run () {
 6.       for (int i=0; i<10000; i++) {
 7.          try {sleep(10);} catch (Exception e) {};
 8.          if (i % 1000 == 0)
 9.             System.out.println(
10.                Thread.currentThread().getName() + ": step "+
i);
11.       }
12.    }
14. }
15.
16. class Main {
17.    public static void main (String argv[]) {
18.       thread2 t2a = new thread2();
19.       thread2 t2b = new thread2();
20.       Thread t = new  Thread(t2a, "thread-2-a");
21.       t.start();
22.       t = new  Thread(t2b,"thread-2-b");
23.       t.start();
24.       for (int i=0; i<10000; i++) {
25.          try {sleep(10);} catch (Exception e) {};
26.          if (i % 1000 == 0)
27.             System.out.println("Main program: step "+ i);
28.       }
29.    }
30. }
```

Στο παράδειγμα αυτό (σε εφαρμογή του πρώτου βασικού βήματος) δημιουργήσαμε τη δική μας κλάση thread2 που υλοποιεί τη διασύνδεση Runnable (γραμμές 3 – 14). Σε αυτή την κλάση τοποθετήσαμε τη μέθοδο run που επιβάλλεται από τη διασύνδεση Runnable και περιέχει τον κώδικα που θα εκτελεί το thread (γραμμές 5 – 12).

Στη συνέχεια δημιουργήσαμε την κλάση Main (γραμμές 16 – 30, στο ρόλο κυρίου προγράμματος) η οποία διαθέτει τη μέθοδο main και ως εκ τούτου είναι εκείνη που

ξεκινάει την εκτέλεση του προγράμματος μας. Αυτή (η μέθοδος main ή αλλιώς το κύριο μας πρόγραμμα) κάνει τρία πράγματα (τρεις ενέργειες). Η πρώτη ενέργεια είναι σύνθετη και υλοποιεί το δεύτερο, τρίτο και τέταρτο βασικό βήμα που αναφέρθηκε παραπάνω. Δηλαδή, δημιουργεί το στιγμιότυπο t2a της κλάσης thread2 (γραμμή 18, πρόκειται για το δεύτερο βασικό βήμα που αναφέρθηκε παραπάνω), στη συνέχεια ξεκινάει ένα στιγμιότυπο της κλάσης Thread και του διαβιβάζει ως παράμετρο το στιγμιότυπο t2a της δικιάς μας κλάσης (στη γραμμή 20, πρόκειται για το τρίτο βασικό βήμα που αναφέρθηκε παραπάνω) και μετά ξεκινάει το στιγμιότυπο της κλάσης Thread (γραμμή 21). Η δεύτερη ενέργεια της main είναι παρόμοια με την πρώτη. Επειδή έχουμε δύο Threads που ξεκινούν από τη main, επαναλαμβάνουμε τα προηγούμενα και δημιουργούμε ένα δεύτερο στιγμιότυπο t2b της κλάσης thread2 (γραμμή 19), στη συνέχεια ξεκινάμε ένα δεύτερο στιγμιότυπο της κλάσης Thread και του διαβιβάζουμε ως παράμετρο το στιγμιότυπο t2b της δικιάς μας κλάσης (στη γραμμή 22) και μετά ξεκινάει το δεύτερο στιγμιότυπο της κλάσης Thread (γραμμή 23). Η τελευταία (τρίτη ενέργεια) που κάνει η main είναι να τρέχει ένα επαναληπτικό σχήμα δέκα χιλιάδων (10000) βημάτων όπου κάθε χίλια βήματα εμφανίζει ένα μήνυμα. Ανάλογη είναι και η εργασία (run) που κάνει και το κάθε στιγμιότυπο της thread2 (thread-2-a και thread-2-b).

Σε αυτό το πρόγραμμα έχουμε 3 thread. Το πρώτο ξεκινάει με την εκτέλεση της main από την JVM. Λίγο μετά (γραμμή 21) ξεκινάει το δεύτερο thread (το thread-2-a) και λίγο πιο μετά (γραμμή 23) ξεκινάει το τρίτο thread (το thread-2-b). Τα thread αυτά, όπως και πριν στην υλοποίηση με επέκταση της κλάσης Thread, ανταγωνίζονται να λάβουν επεξεργαστικούς κύκλους.

Διαφορές – Σύγκριση

Είχαμε πει (για το τρίτο βασικό βήμα) ότι στον κατασκευαστή της κλάσης Thread περνάμε ως παράμετρο το στιγμιότυπο της δικιάς μας κλάσης. Αν είστε παρατηρητικοί θα έχετε δει ότι στις γραμμές 20 και 22 χρησιμοποιήσαμε διπλό όρισμα στην κλήση του κατασκευαστή της κλάσης Thread. Το πρώτο όρισμα ήταν πράγματι το στιγμιότυπο της δικιάς μας κλάσης. Το ερώτημα είναι: επιτρέπεται το δεύτερο όρισμα και ποιος είναι ο ρόλος του; Η απάντηση είναι καταφατική (βλέπε πολυμορφισμός) και το δεύτερο όρισμα είναι για να λαμβάνει το thread το όνομα που εμείς επιθυμούμε. Αυτό που στην δημιουργία νήματος ως επέκταση της κλάσης Thread γινότανε με τον κατασκευαστή της κλάσης, εδώ γίνεται κατά τη δημιουργία του στιγμιότυπου της κλάσης Thread.

Αν αναρωτιέστε ποια από τις δύο δυνατότητες για τη δημιουργία νήματος είναι καλύτερα να χρησιμοποιούμε, η απάντηση είναι προτιμήστε τη δημιουργία νήματος ως κλάση που υλοποιεί το interface Runnable. Με αυτό τον τρόπο μπορείτε να

δημιουργήσετε το δικό σας Νήμα το οποίο θα είναι επέκταση και θα έχει τη λειτουργικότητα οποιασδήποτε άλλης κλάσης.

Μια άλλη διαφορά που συναντήσαμε στα παραδείγματα των δύο μεθόδων δημιουργίας thread έχει να κάνει με τον τρόπο που ανακαλέσαμε (εντός της run) το όνομα του thread. Στο πρώτο παράδειγμα, γραμμή 13, χρησιμοποιήσαμε this.getName(). Στο δεύτερο παράδειγμα, γραμμή 10, χρησιμοποιήσαμε Thread.currentThread().getName(). Αυτό έχει απλή εξήγηση. Στην πρώτη περίπτωση έχουμε μια κλάση που επεκτείνει την κλάση Thread και επομένως έχει διαθέσιμες όλες τις μεθόδους της κλάσης Thread. Στην δεύτερη περίπτωση η κλάση που δημιουργούμε δεν βασίζεται στην (δεν κληρονομεί την) Thread και επομένως χρησιμοποιούμε τις στατικές μεθόδους (όπως είναι η currentThread) της κλάσης Thread που είναι διαθέσιμες προς όλους.