# Υ325 - Αντικειμενοστρεφής Τεχνολογία

2021

Τμ. ΗΜ&ΤΥ - Πανεπιστήμιο Πατρών

# Εβδομάδα 11 - 23/12/2021

# Java Threads

# Δημιουργία και εκτέλεση Thread

Ένα thread στη Java είναι ένα αντικείμενο, σαν όλα τα άλλα αντικείμενα της Java, και είναι στιγμιότυπο της κλάσης `java.lang.Thread`.

Ένα στιγμιότυπο τύπου `Thread` μπορεί να εκτελέσει κώδικα σε ένα ξεχωριστό νήμα.

```
    //δημιουργία στιγμιοτύπου της κλάσης Thread
    Thread thread = new Thread();
    //εκκίνηση του νήματος
    thread.start();
```

# Δημιουργία και εκτέλεση Thread #2

Έχουμε δύο τρόπους για να γράψουμε κώδικα που θα εκτελεστεί σε ξεχωριστό νήμα.

Α. να κληρονομήσουμε την κλάση `java.lang.Thread`:

```
public class DummyThread extends Thread {
    public void run(){
        System.out.println("Working...");
        //εδώ κώδικας που κάνει τη δουλειά...
    }
}

...

DummyThread dThread = new DummyThread();
//η start() ξεκινά τον κώδικα που περιγράφεται στη run()
//αλλά δεν περιμένει να τελειώσει η run().
dthread.start();
```

# Δημιουργία και εκτέλεση Thread #3

Θα μπορούσαμε να κατασκευάσουμε και ανώνυμη υποκλάση τύπου `Thread`:

```
    Thread myThread = new Thread() {
        public void run() {
                System.out.println("Working...");
                //εδώ κώδικας που κάνει τη δουλειά...
            }
        }

    myThread.start();
```

# Δημιουργία και εκτέλεση Thread #4

Β. Να υλοποιήσουμε το `java.lang.Runnable` interface σε μια δική μας κλάση, και να περάσουμε ένα στιγμιότυπό της σε ένα `Thead`:

```
public class MyRunnable implements Runnable {
    public void run(){
        System.out.println("Working...");
        //εδώ κώδικας που κάνει τη δουλειά...
    }
}

...

Thread dThread = new Thread(new MyRunnable());
dthread.start();
```

# Δημιουργία και εκτέλεση Thread #5

Μπορούμε να χρησιμοποιήσουμε και τους δύο τρόπους εξίσου. Η υλοποίηση του `Runnable` προσφέρει τα πλεονεκτήματα της πολλαπλής κληρονομικότητας που έχουμε με το μηχανισμό του `interface`.

Προσοχή, στη διαφορά όμως: Για να ξεκινήσουμε ένα νήμα καλούμε τη μέθοδο `start()`. Αν καλέσουμε τη `run()` ο κώδικας θα εκτελεστεί στο τρέχον νήμα, δε θα δημιουργηθεί καινούριο.

# Παύση ενός νήματος

Η εκτέλεση του νήματος μπορεί να παυθεί (χωρίς να τερματίσει) καλώντας της στατική μέθοδο `sleep()`.

```
...
run() {
    ...
    Thread.sleep(1000);
    ...
}
```

Το νήμα θα “κοιμηθεί” για περίπου 1000ms, ίσως και περισσότερα.

# Κρίσιμη περιοχή

Η κρίσιμη περιοχή (critical region) είναι το τμήμα του κώδικα που μπορεί να εκτελεστεί από πολλά νήματα και που η σειρά με την οποία τα νήματα θα την εκτελέσουν έχει σημασία για το αποτέλεσμα.

Π.χ. δύο νήματα μπορεί να αλλάζουν την ίδια μεταβλητή:

```
public class Counter {
    private int count = 0;
    public void increase() {
        this.count += 1;
    }
}
```

Η εκτέλεση της `increase()` από ένα νήμα μπορεί να διακοπεί στη μέση και να δοθεί ο έλεγχος σε άλλο νήμα που εκτελεί την ίδια μέθοδο.

# Κρίσιμη περιοχή #2

Μια πιθανή σειρά εκτέλεσης είναι:

| Νήμα | Α | Β |
|---|---|---|
| Α | διαβάζει το 0 στον καταχ. | |
| Β | | διαβάζει το 0 στον καταχ. |
| Β | | αυξάνει +1 |
| Β | | αποθηκεύει στη μνήμη 1 |
| Α | αυξάνει +1 | |
| Α | αποθηκεύει στη μνήμη 1 | |

Το αποτέλεσμα θα είναι 1, ενώ θα θέλαμε να είναι 2. Η σειρά της εκτέλεσης δεν ξέρουμε ποια θα είναι, εξαρτάται από τον scheduler.

# Η λέξη-κλειδή synchronized

Ο μηχανισμός που επιτρέπει να ορίσουμε ένα τμήμα κώδικα ως `syncronized` ήταν ο πρώτος που παρείχε η Java για να εξασφαλίσουμε την αποκλειστική πρόσβαση σε μνήμη που είναι προσβάσιμη από πολλά νήματα.

Ένα τμήμα κώδικα που έχει σημειωθεί ως `synchronized` μπορεί να εκτελεστεί μόνο από ένα νήμα κάθε φορά. Αν ένα δεύτερο νήμα προσπαθήσει να εισέλθει σε ένα τμήμα `synchronized`, θα εμποδιστεί (block) μέχρις ώτου το νήμα που εκτελεί τον κώδικα εξέλθει από το τμήμα που είναι `synchronized`.

# Η λέξη-κλειδή synchronized #2

Σε κάθε στιγμιότυπο της `Counter`, μόνο ένα νήμα μπορεί να εκτελέσει οποιαδήποτε από τις δύο μεθόδους που είναι `synchronized`. Π.χ. όταν εκτελείται η `increase()` σε ένα νήμα, κανένα άλλο νήμα δεν μπορεί να την εκτελέσει, αλλά ούτε και την `decrease()`.

```
public class Counter {
    private int count = 0;

    synchronized public void increase() {
        this.count += 1;
    }
    synchronized public void decrease() {
        this.count -= 1;
    }
}
```

Αυτό ισχύει για κάθε στιγμιότυπο, αν έχουμε π.χ. ένα δεύτερο στιγμιότυπο της `Counter`, αυτό θα είναι ανεξάρτητο.