

Προηγμένα Θέματα Αντικειμενοστραφούς Προγραμματισμού (Java)

-----

ΑΤΟΜΙΚΗ ΕΡΓΑΣΙΑ

ΦΟΙΤΗΤΗΣ: ΠΑΝΑΓΙΩΤΗΣ ΚΟΝΤΟΣ

ΑΜ: mpsp2215

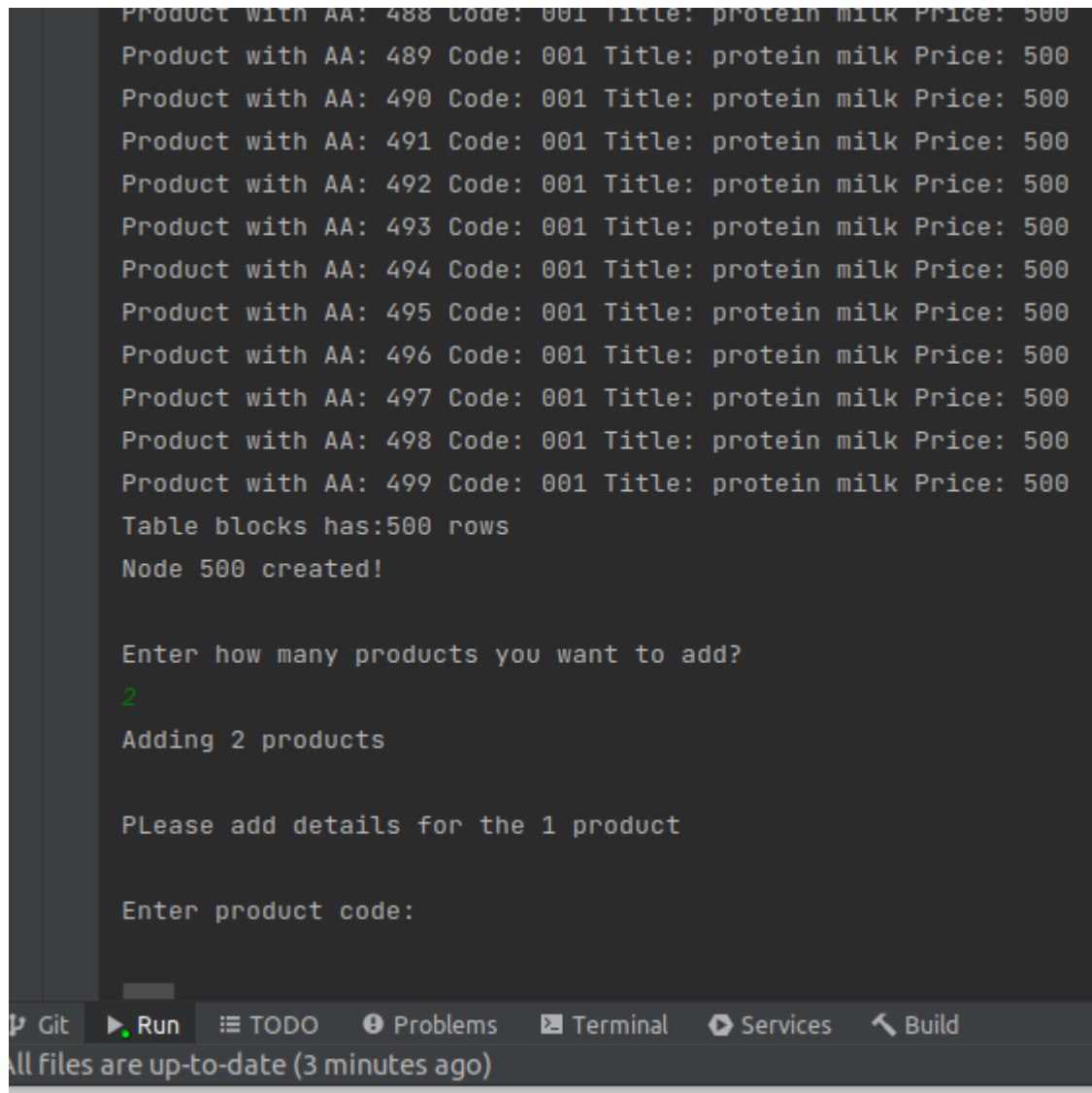
## Εισαγωγή

Η παρακατω εργασία αφορά την αναπτυξη μια εφαρμογης Blockchain με Java οπου γινεται χρηση του Threading και benchmarking των αποτελεσμάτων. Η εφαρμογη παρουσιαζει 3 εκδοχες η 1η χωρις την χρηση νηματων, η 2η με την βοηθεια των thread pools και η τελευταία με την δημιουργια δικων μας νηματων. Απο την συγκριση τους θα δουμε τελικα πως η χρηση των thread pools αποδειχτηκε και η καλυτερη και και πιο αποδοτικη επιλογη.

### Screenshot απο εκτέλεση της εφαρμογής

Αρχικα η εφαρμογη διαβαζει τα προϊόντα που υπάρχουν σε μια **file-based** βάση δεδομένων και αφου φορτώσει ολο το υπαρχον blockchain μας ζηται τον αριθμο των προϊόντων που θα θελουμε προσθεσουμε σε αυτο. (Στην περιπτωση αδειας βασης ξεκιναι απο το Genesis Block)

Αφου δωσουμε αυτον τον αριθμο πρεπει να δωσουμε τα χαρακτηριστικα για το καθε προιον



```
Product with AA: 488 Code: 001 Title: protein milk Price: 500
Product with AA: 489 Code: 001 Title: protein milk Price: 500
Product with AA: 490 Code: 001 Title: protein milk Price: 500
Product with AA: 491 Code: 001 Title: protein milk Price: 500
Product with AA: 492 Code: 001 Title: protein milk Price: 500
Product with AA: 493 Code: 001 Title: protein milk Price: 500
Product with AA: 494 Code: 001 Title: protein milk Price: 500
Product with AA: 495 Code: 001 Title: protein milk Price: 500
Product with AA: 496 Code: 001 Title: protein milk Price: 500
Product with AA: 497 Code: 001 Title: protein milk Price: 500
Product with AA: 498 Code: 001 Title: protein milk Price: 500
Product with AA: 499 Code: 001 Title: protein milk Price: 500
Table blocks has:500 rows
Node 500 created!

Enter how many products you want to add?
2
Adding 2 products

Please add details for the 1 product

Enter product code:
```

Αφου δώσουμε τα χαρακτηριστικά των προϊόντων, ψαχνει για να δει αν το προϊόν υπάρχει ήδη στην βάση, αν υπάρχει τότε ανεβαζει το AA +1 αλλιως ξεκιναι με 1. Η εφαρμογή για κάθε προϊόν που προσθέτουμε κανει mine ενα καινούργιο block και το συνδεει με τα προηγούμενα και υστερα το **προσθετει** στην file-based βάση δεδομένων.

Επειτα αφου προσθεσουμε τα προϊόντα μας υπάρχει η δυνατότητα του search μέσα στο blockchain συγκεκριμένα μπορείς να βρεις να κανεις search με 2 τρόπους.

1. Ειτε κανεις multisearch **by name** οπου απλα δινεις ενα substring που μπορεί να είναι το ονομα του προϊόντος ειτε μερικα γραμματα απο το ονομα και σου επιστρέφει όλα τα προϊόντα που περιέχουν αυτο το substring.

```
asdasd
Product doesn't exist
Node 506 created!

Are any products you want to search?
Search product
  1. by name
  2. by code
  3. don't want to search
1
Enter name
pasta
Product with AA: 1 Code: 45 Title: pasta-4-seasons Price: 23
Product with AA: 1 Code: 21 Title: pasta-napolitana Price: 21
Product with AA: 1 Code: 34 Title: pasta Price: 21
Product with AA: 1 Code: 98 Title: pasta Price: 12

Do you want to see data about a product?
1. Yes
2. No

Run | TODO | Problems | Terminal | Services | Build
Completed successfully in 693 ms (a minute ago)
```

2. είτε **by product code** οπου επιλέγεις αν θες να δεις **το πιο παλιο ή το πιο καινούργιο block** δηλαδή την 1η ή την τελευταία τιμή αυτού του προϊόντος.

```
Adding 0 products
```

```
Are any products you want to search?
```

```
Search product
```

1. by name
2. by code
3. don't want to search

```
1
```

```
Enter code
```

```
98
```

```
Do you want to search for
```

1. Oldest panos\_App\_1.Block
2. Newest panos\_App\_1.Block

```
2
```

```
panos_App_1.Block@dc24521
```

```
Product with AA: 1 Code: 98 Title: pasta Price: 12
```

```
Do you want to see data about a product?
```

```
1. Yes
```

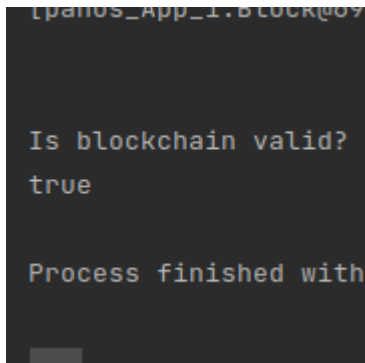
```
2. No
```

Στην συνέχεια μας ζητείται θέλουμε να δουμε την **εξελιξη της τιμης για καποιο προιον** σε μορφη πινακα. Οπου απλα εισάγοντας τον product code μας εμφανίζονται σαν πινακας τα προϊόντα η ημερομηνίες εισαγωγης και η τιμή τους.

```
Do you want to see data about a product?
1. Yes
2. No
1
Enter product code
001
```

Product	Date	Price
protein milk	Sat Jan 07 18:27:00 EET 2023	500\$
protein milk	Sat Jan 07 18:29:05 EET 2023	500\$
protein milk	Sat Jan 07 18:31:10 EET 2023	500\$
protein milk	Sat Jan 07 18:33:14 EET 2023	500\$
protein milk	Sat Jan 07 18:35:18 EET 2023	500\$
protein milk	Sat Jan 07 18:42:19 EET 2023	500\$
protein milk	Sat Jan 07 18:42:20 EET 2023	500\$
protein milk	Sat Jan 07 18:42:21 EET 2023	500\$
protein milk	Sat Jan 07 18:42:23 EET 2023	500\$
protein milk	Sat Jan 07 18:42:24 EET 2023	500\$
protein milk	Sat Jan 07 18:42:25 EET 2023	500\$
protein milk	Sat Jan 07 18:42:26 EET 2023	500\$

Τέλος, η εφαρμογή ελέγχει κάθε φορά αν το blockchain είναι valid. Επικυρώνοντας έτσι το Σύνολο των Blocks



```
[panos_App_1.Block@07  
  
Is blockchain valid?  
true  
  
Process finished with
```

## 2η Έκδοση

### Εφαρμογή των ThreadPools

Στην 2η έκδοση της εφαρμογής έγινε χρήση των threads σε 4 σημεία. Αξίζει να σημειωθεί ότι σε κάθε εφαρμογή τους γινόταν ένα μικρό benchmarking για να βρεθεί ο βέλτιστος αριθμός threads που πρέπει να χρησιμοποιηθούν.

Οι εφαρμογές ήταν κυρίως σε σημεία τα οποία θεωρήθηκαν υπολογιστικά πιο χρονοβόρα ή σε σημεία τα οποία γίνονται πιο χρονοβόρα όσο ο αριθμός των blocks αυξάνεται. Τα σημεία αυτά ήταν:

- **Block Mining Thread** - θεωρητικά το πιο βαρύ task όσο ο αριθμός του blockchain αυξάνεται, βέλτιστος αριθμός threads φάνηκε να ήταν 4.
- **Search By Code Thread** - όσο ο αριθμός του blockchain αυξάνεται, τα search γίνονται πιο αργά, βέλτιστος αριθμός threads φάνηκε να είναι τα 30.
- **Search By Name Thread** - η ίδια λογική με το παραπάνω, βέλτιστος αριθμός threads και εδώ φάνηκε να είναι τα 30.
- **Validate Blockchain Thread** - πρέπει να γίνει iterate και έλεγχος σε όλο το μήκος του blockchain επομένως υπάρχει νόημα για εφαρμογή threads. Βέλτιστος αριθμός threads φάνηκε να είναι τα 30

Η εφαρμογή κινείται στην ίδια λογική με την 1η έκδοση απλά έχουν χρησιμοποιηθεί threads σε αυτά τα 4 σημεία.

### 3η Εκδοση

#### Εφαρμογή των Δικών μας Thread

Στην 3η εκδοση της εφαρμογης εγινε χρηση των δικων μας thread χωρις καποια εξωτερικη βοηθεια. Το task αποδείχτηκε αρκετα challenging καθώς μπορεσε να γινει εφαρμογη threads σε επιπεδο κώδικα μονο σε 2 σημεια. Αξιζει να σημειωθει οτι σε καθε εφαρμογη τους γινotan ενα μικρο benchmarking για να βρεθει ο βελτιστος αριθμος threads που πρεπει να χρησιμοποιηθούν.

Τα σημεία αυτα ηταν:

- **Search By Code Thread** - οσο ο αριθμος του blockchain αυξανεται, τα search γινονται πιο αργα, βελτιστος αριθμος threads φανηκε να ειναι τα 30.
- **Search By Name Thread** -η ιδια λογικη με το παραπάνω, βελτιστος αριθμος threads και εδω φανηκε να ειναι τα 30.

Επιπλέον ειναι σημαντικό να τονισθεί οτι εγινε προσπάθεια για εφαρμογή τους σε περισσοτερα σημεια(Validate Blockchain Thread,Block Mining Thread) αλλα λογω περιορισμένου χρονου δεν καταφέρθηκε να γινει επιτυχημένα.

## Σύγκριση Αποτελεσμάτων

Παρακάτω φαίνονται screenshot απο τα αποτελέσματα του benchmarking για τις 3 εκδοσεις της εφαρμογης. Συγκεκριμένα, γινεται benchmarking για throughput και average time καθώς και παρατηρησεις για τον αριθμο προϊόντων που δημιουργήθηκαν στην βαση.

### Throughput Testing - 1η εκδοση(No threads)

Score = 0.324 ops/s

```
Result "benchmark.Main.test_apps":
  0.324 ±(99.9%) 0.002 ops/s [Average]
  (min, avg, max) = (0.318, 0.324, 0.329), stdev = 0.003
  CI (99.9%): [0.322, 0.327] (assumes normal distribution)

# Run complete. Total time: 00:10:20

REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.

NOTE: Current JVM experimentally supports Compiler Blackholes, and they are in use. Please exercise
extra caution when trusting the results, look into the generated code to check the benchmark still
works, and factor in a small probability of new VM bugs. Additionally, while comparisons between
different JVMs are already problematic, the performance difference caused by different Blackhole
modes can be very significant. Please make sure you use the consistent Blackhole mode for comparisons.

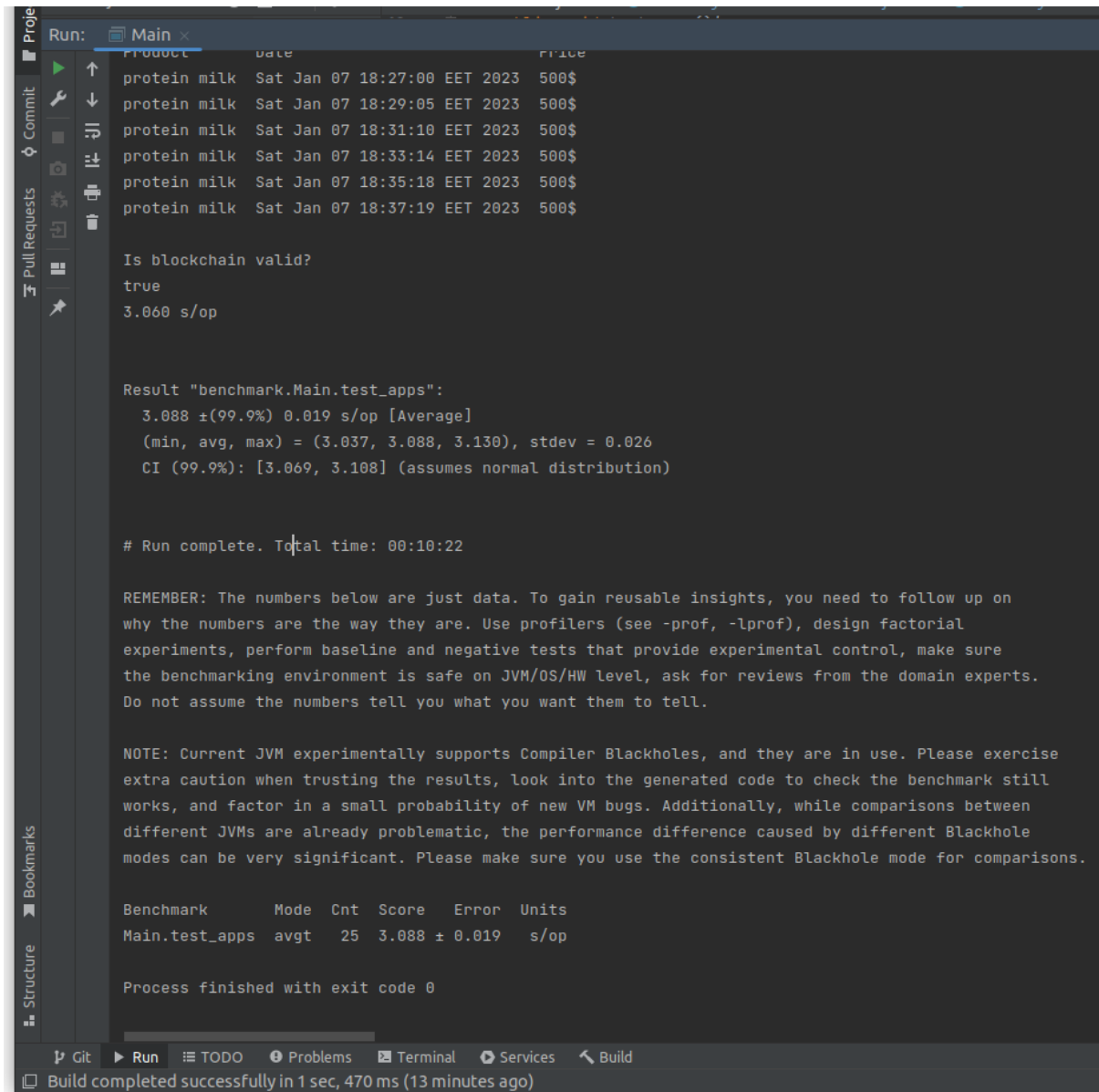
Benchmark      Mode  Cnt   Score   Error  Units
Main.test_apps  thrpt   25  0.324 ± 0.002  ops/s

Process finished with exit code 0
```



## Average Time Testing - 1η εκδοση(No threads)

Score = 3.088 s/op



The screenshot shows an IDE terminal window with the following content:

```
Run: Main x
Product      Date      Price
protein milk Sat Jan 07 18:27:00 EET 2023 500$
protein milk Sat Jan 07 18:29:05 EET 2023 500$
protein milk Sat Jan 07 18:31:10 EET 2023 500$
protein milk Sat Jan 07 18:33:14 EET 2023 500$
protein milk Sat Jan 07 18:35:18 EET 2023 500$
protein milk Sat Jan 07 18:37:19 EET 2023 500$

Is blockchain valid?
true
3.060 s/op

Result "benchmark.Main.test_apps":
3.088 ±(99.9%) 0.019 s/op [Average]
(min, avg, max) = (3.037, 3.088, 3.130), stdev = 0.026
CI (99.9%): [3.069, 3.108] (assumes normal distribution)

# Run complete. Total time: 00:10:22

REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.

NOTE: Current JVM experimentally supports Compiler Blackholes, and they are in use. Please exercise
extra caution when trusting the results, look into the generated code to check the benchmark still
works, and factor in a small probability of new VM bugs. Additionally, while comparisons between
different JVMs are already problematic, the performance difference caused by different Blackhole
modes can be very significant. Please make sure you use the consistent Blackhole mode for comparisons.

Benchmark      Mode  Cnt  Score   Error  Units
Main.test_apps  avgt   25  3.088 ± 0.019  s/op

Process finished with exit code 0
```

At the bottom of the terminal, there is a status bar that reads: "Build completed successfully in 1 sec, 470 ms (13 minutes ago)".

## Throughput Testing - 2η εκδοση(ThreadPools)

Score = 0.920 ops/s

```
protein milk Thu Jan 05 01:41:10 EET 2023 500$
protein milk Thu Jan 05 01:41:11 EET 2023 500$
protein milk Thu Jan 05 01:41:13 EET 2023 500$
protein milk Thu Jan 05 01:41:14 EET 2023 500$
protein milk Thu Jan 05 01:41:15 EET 2023 500$
protein milk Thu Jan 05 01:41:16 EET 2023 500$

Is blockchain valid?
true
0.942 ops/s
|

Result "benchmark.Main.test_apps":
  0.920 ±(99.9%) 0.024 ops/s [Average]
  (min, avg, max) = (0.800, 0.920, 0.949), stdev = 0.032
  CI (99.9%): [0.896, 0.943] (assumes normal distribution)

# Run complete. Total time: 00:08:59

REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.

NOTE: Current JVM experimentally supports Compiler Blackholes, and they are in use. Please exercise
extra caution when trusting the results, look into the generated code to check the benchmark still
works, and factor in a small probability of new VM bugs. Additionally, while comparisons between
different JVMs are already problematic, the performance difference caused by different Blackhole
modes can be very significant. Please make sure you use the consistent Blackhole mode for comparisons.

Benchmark      Mode  Cnt  Score   Error  Units
Main.test_apps  thrpt   25  0.920 ± 0.024  ops/s

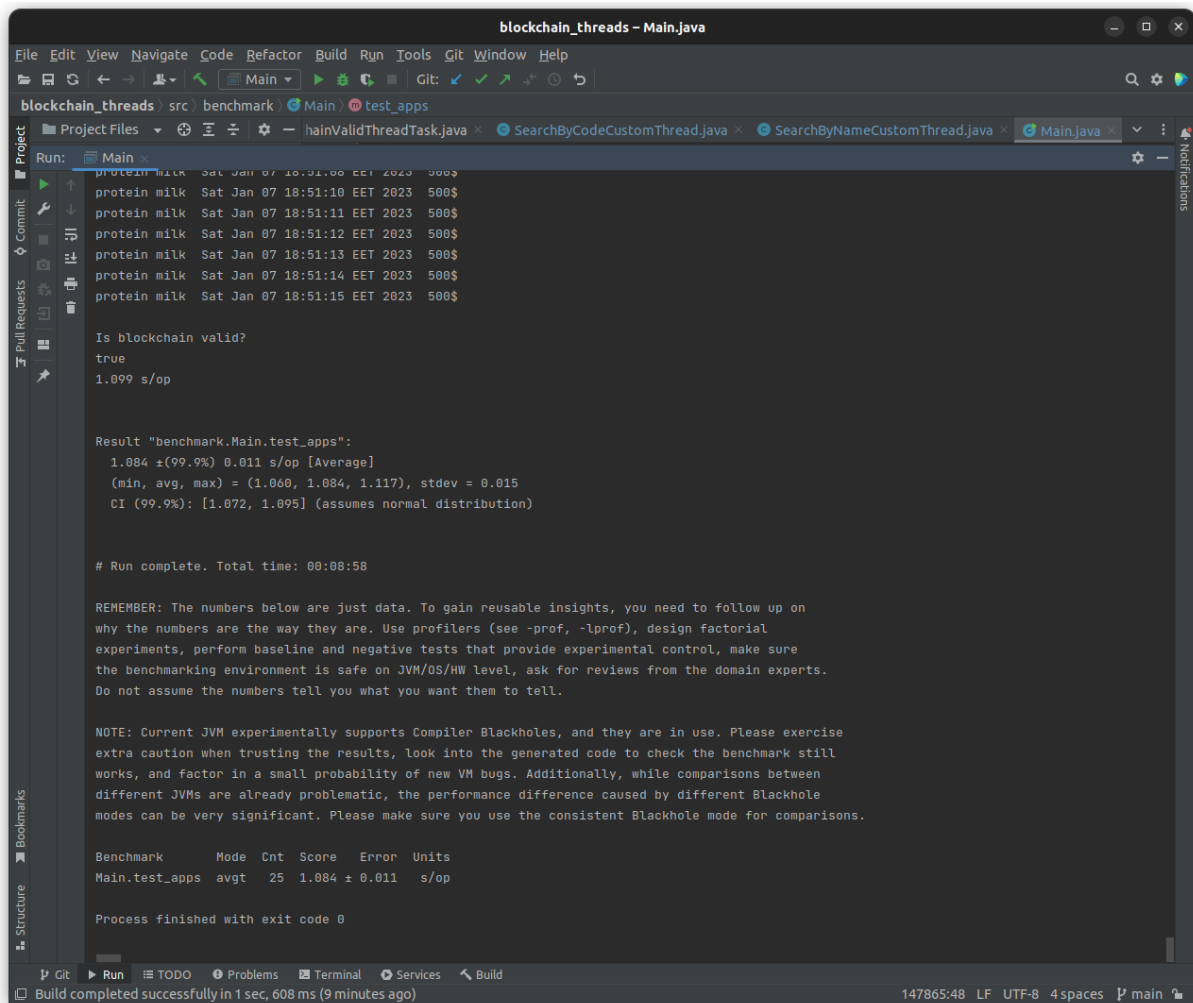
Process finished with exit code 0
```

Run | TODO | Problems | Terminal | Services | Build

Completed successfully in 1 sec 453 ms (11 minutes ago)

## Average Time Testing - 2η εκδοση(ThreadPools)

Score = 1.004 s/op



```
blockchain_threads - Main.java
File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Project Files src benchmark Main test_apps
Run: Main x
protein milk Sat Jan 07 18:51:08 EET 2023 500$
protein milk Sat Jan 07 18:51:10 EET 2023 500$
protein milk Sat Jan 07 18:51:11 EET 2023 500$
protein milk Sat Jan 07 18:51:12 EET 2023 500$
protein milk Sat Jan 07 18:51:13 EET 2023 500$
protein milk Sat Jan 07 18:51:14 EET 2023 500$
protein milk Sat Jan 07 18:51:15 EET 2023 500$

Is blockchain valid?
true
1.099 s/op

Result "benchmark.Main.test_apps":
1.084 ±(99.9%) 0.011 s/op [Average]
(min, avg, max) = (1.060, 1.084, 1.117), stdev = 0.015
CI (99.9%): [1.072, 1.095] (assumes normal distribution)

# Run complete. Total time: 00:08:58

REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.

NOTE: Current JVM experimentally supports Compiler Blackholes, and they are in use. Please exercise
extra caution when trusting the results, look into the generated code to check the benchmark still
works, and factor in a small probability of new VM bugs. Additionally, while comparisons between
different JVMs are already problematic, the performance difference caused by different Blackhole
modes can be very significant. Please make sure you use the consistent Blackhole mode for comparisons.

Benchmark      Mode  Cnt  Score   Error  Units
Main.test_apps  avgt   25  1.084 ± 0.011  s/op

Process finished with exit code 0

Git Run TODO Problems Terminal Services Build
Build completed successfully in 1 sec, 608 ms (9 minutes ago) 147865:48 LF UTF-8 4spaces main
```

## Throughput Testing - 3η εκδοχή(MyThread)

Score = ( Τρέχει για 4 seconds και δίνει Error )

```
-----
java.lang.InterruptedException Create breakpoint
    at java.base/java.util.concurrent.locks.ReentrantLock$Sync.lockInterruptibly(ReentrantLock.java:159)
    at java.base/java.util.concurrent.locks.ReentrantLock.lockInterruptibly(ReentrantLock.java:372)
    at java.base/java.util.concurrent.LinkedBlockingQueue.poll(LinkedBlockingQueue.java:455)
    at java.base/java.util.concurrent.ExecutorCompletionService.poll(ExecutorCompletionService.java:209)
    at org.openjdk.jmh.runner.BenchmarkHandler.runIteration(BenchmarkHandler.java:368)
    at org.openjdk.jmh.runner.BaseRunner.runBenchmark(BaseRunner.java:261)
    at org.openjdk.jmh.runner.BaseRunner.runBenchmark(BaseRunner.java:233)
    at org.openjdk.jmh.runner.BaseRunner.doSingle(BaseRunner.java:138)
    at org.openjdk.jmh.runner.BaseRunner.runBenchmarksForked(BaseRunner.java:75)
    at org.openjdk.jmh.runner.ForkedRunner.run(ForkedRunner.java:72)
    at org.openjdk.jmh.runner.ForkedMain.main(ForkedMain.java:86)

# Run complete. Total time: 00:00:05

REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.

NOTE: Current JVM experimentally supports Compiler Blackholes, and they are in use. Please exercise
extra caution when trusting the results, look into the generated code to check the benchmark still
works, and factor in a small probability of new VM bugs. Additionally, while comparisons between
different JVMs are already problematic, the performance difference caused by different Blackhole
modes can be very significant. Please make sure you use the consistent Blackhole mode for comparisons.

Benchmark Mode Cnt Score Error Units

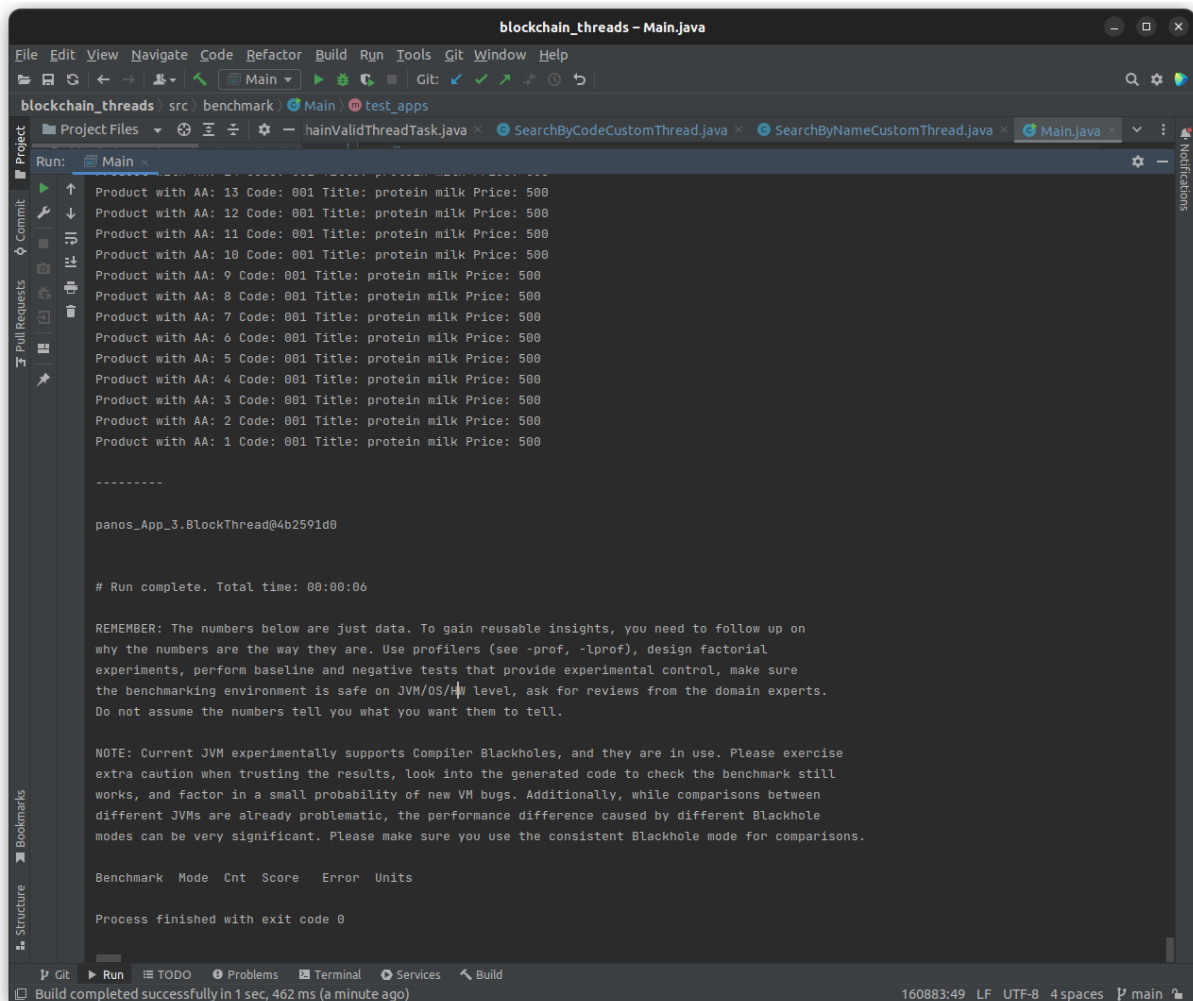
Process finished with exit code 0
|
```

Run | TODO | Problems | Terminal | Services | Build

completed successfully in 1 sec, 321 ms (a minute ago)

## Average Time Testing - 3η εκδοση(MyThread)

Score = ( Τρέχει για 4 seconds και δίνει Error )



```
blockchain_threads - Main.java
File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Main
blockchain_threads | src | benchmark | Main | test_apps
Project Files | hainValidThreadTask.java | SearchByCodeCustomThread.java | SearchByNameCustomThread.java | Main.java
Run: Main
Product with AA: 13 Code: 001 Title: protein milk Price: 500
Product with AA: 12 Code: 001 Title: protein milk Price: 500
Product with AA: 11 Code: 001 Title: protein milk Price: 500
Product with AA: 10 Code: 001 Title: protein milk Price: 500
Product with AA: 9 Code: 001 Title: protein milk Price: 500
Product with AA: 8 Code: 001 Title: protein milk Price: 500
Product with AA: 7 Code: 001 Title: protein milk Price: 500
Product with AA: 6 Code: 001 Title: protein milk Price: 500
Product with AA: 5 Code: 001 Title: protein milk Price: 500
Product with AA: 4 Code: 001 Title: protein milk Price: 500
Product with AA: 3 Code: 001 Title: protein milk Price: 500
Product with AA: 2 Code: 001 Title: protein milk Price: 500
Product with AA: 1 Code: 001 Title: protein milk Price: 500

-----

panos_App_3.BlockThread@4b2591d0

# Run complete. Total time: 00:00:06

REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.

NOTE: Current JVM experimentally supports Compiler Blackholes, and they are in use. Please exercise
extra caution when trusting the results, look into the generated code to check the benchmark still
works, and factor in a small probability of new VM bugs. Additionally, while comparisons between
different JVMs are already problematic, the performance difference caused by different Blackhole
modes can be very significant. Please make sure you use the consistent Blackhole mode for comparisons.

Benchmark Mode Cnt Score Error Units

Process finished with exit code 0

Git Run TODO Problems Terminal Services Build
Build completed successfully in 1 sec, 462 ms (a minute ago) 160883:49 LF UTF-8 4spaces main
```

### Αποτελέσματα - Συμπεράσματα

Κατα το benchmarking στην εφαρμογή μας αντι να παίρνουμε τα προϊόντα και τα δεδομένα τους σαν input τα δίνουμε εμείς απο πριν. Και καθε φορα που ετρεχε η εφαρμογή εφτιαχνε και 1 προϊόν στην βαση.

	Throughput	Average Time	Blocks Created
No threads	0.324 ops/s	3.088 s/op	10
ThreadPools	0.920 ops/s	1.004 s/op	492
My own threads	Error	Error	4

Οπως φαινεται απο τα αποτελεσματα εγινε **επιτυχημενη χρηση των thread pools** καθως σχεδον **τριπλασιαστηκε το throughput** αλλα και **υποτριπλασιαστηκε ο μεσος χρονος** της εφαρμογης.

Επειτα,ειναι σημαντικο να σημειωθει οτι τα συνεχομενα iteration που ετρεξε το benchmarking φανηκε να δημιουργουν συνεχομενα προϊόντα στην βαση και να μην κολλαει πουθενα ο server δηλαδη να αυξανεται το responsiveness και να **διαχειρίζεται επιτυχημένα το load**, ενω αντιθετα οταν δεν γινοταν χρηση των threads ο server πολλες φορες φανηκε να κολλαει πριν φτιαξει καποιο προιον καθως δεν μπορουσε να κανει αποτελεσματικο load balancing.

Η **εφαρμογη των δικων μας νηματων** ενω φανηκε να εγινε επιτυχημένα οταν τρέχαμε αρχικα την εφαρμογη manually οταν δοκιμασαμε να κανουμε το benchmarking τα threads παρουσιασαν errors αποδεικνυοντας οτι το να εχεις μια ασφαλη και γρηγορη εφαρμογη των νηματων χρησημοποιώντας δικα σου threads ειναι μια αρκετα προκλητικη και χρονοβορα διαδικασια .

Συνοπτικα απο την συγκεκριμένη εφαρμογή φανηκε οτι,  
τα **πλεονεκτηματα** της χρησης των threads ειναι:

- **Βελτιομένη Απόδοση** - τρεχοντας πολλαπλα tasks ασυγχρονα φανηκε να βελτιωνεται η αποδοση.
- **Responsiveness** - Το προγραμμα μπορουσε να εκτελει tasks στο background χωρις να επηρεαζει την main εφαρμογη, το οποιο είχε πλεονεκτήματα στο load balancing.
- **Concurrency** - μας βοηθησε στο να τρεχουμε πολλαπλα tasks παραλληλα τα οποια ειναι ανεξαρτητα μεταξυ τους.

Αντιθετα τα **μειονεκτήματα** φανηκε να είναι:

- **Πολυπλοκότητα** - η χρήση των νημάτων προσθεσε πολυπλοκοτητα αρχικα στην συγγραφη του κωδικα αλλα και στην διαχειριση, την επικοινωνία και συγχρονισμο των νημάτων μεταξύ τους.
- **Deadlocks** - οταν δεν γινεται σωστη χρηση των νημάτων 2 νηματα περιμενουν ταυτοχρονα το ενα το αλλο με αποτελεσμα το προγραμμα να παγωσει.
- **Debugging** - Η δυσκολια στην διαχειριση του λογισμικου και στην διορθωση του φαινεται να αποτελει χρονοβορα διαδικασια.

Παρολα αυτα με την χρηση των **thread pools** μπορουμε να αποφυγουμε πολλα απο αυτα τα μειονεκτήματα καθως **μας βοήθησαν στην:**

- **Απλούστευση στην διαχειριση και επικοινωνία των νημάτων** καθως μας δινουν ενα υψηλοτερο επιπεδο abstraction και το οποιο διευκολύνει την συγγραφη και την διαχειριση του κωδικα.
- **Καλυτερη και πιο Ασφαλες Χρηση των Resources** - εχουμε ενα συγκεκριμενο αριθμο threads που μπορει να τρεχει ταυτοχρονα στο συστημα μας αλλα και μπορουμε να επαναχρησιμοποιούμε threads και να βελτιώνουμε την αποδοση της εφαρμογης μας.
- **Ευκολοτερο Debugging** - το υψηλοτερο επιπεδο abstraction μας δινει λιγοτερο flexibility και συνεπως ευκολοτερο debugging

### Συμπερασματα

Συμπερασματικα, η καλυτερη εκδοση του Blockchain φαινεται απο ολες τις αποψεις οτι ειναι αυτη με τα thread pools καθως βελτιωνει σημαντικα οτι εκανε η 1η εκδοση χωρις ομως να παρουσιαζει μεγαλες δυσκολιες που ερχονται με την χρηση νημάτων στην συγγραφη,την διαχειριση και το debugging του κωδικα. Ενω, αντιθετα η χειρότερη φαινεται να ειναι η 3η εκδοση καθως ακομα και αν καταφερουμε να δημιουργησουμε πετυχημενα τον κωδικα για τα threads, η διαχειριση και το debugging του θα ειναι κατι που σιγουρα θα μας δημιουργησει προκλησεις στο μελλον.