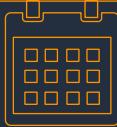


Ενότητα 2

Διαχείριση Μνήμης και Σχεδιασμός Κλάσεων

Ενότητα 2: Διαχείριση Μνήμης και Σχεδιασμός Κλάσεων

Οργάνωση και Διαχείριση Μνήμης



Ενότητα 2: Διαχείριση Μνήμης και Σχεδιασμός Κλάσεων

Μάθημα 10: 23/2/2024

Οργάνωση μνήμης Η/Υ
Οργάνωση μνήμης JVM
Κατασκευή αντικειμένων

Περίγραμμα

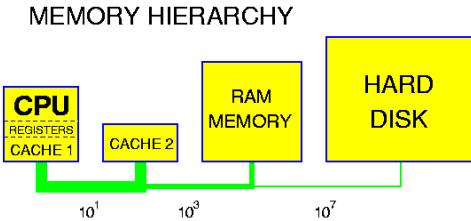


- Οργάνωση και Διαχείριση Μνήμης
- Διεργασίες και Εικονική Μνήμη
- Διαχείριση Μνήμης Java κ. JVM
- Αρχικοποιήσεις και Κατασκευή Αντικειμένων
- Σκύβαλα και Αποκομιδή
- Στατικές Μεταβλητές και Μέθοδοι
- Περιβάλλουσες Κλάσεις
- Παράμετροι Κλάσεων
- Έλεγχος ισότητας αντικειμένων
- Αναλλοίωτοι Περιορισμοί
- Παραβίαση ιδιωτικότητας και κατασκευαστές αντιγράφου



ΠΟΥ ΑΠΟΘΗΚΕΥΟΝΤΑΙ ΤΑ ΔΕΔΟΜΕΝΑ ΤΩΝ ΠΡΟΓΡΑΜΜΑΤΩΝ ΜΑΣ;

Οργάνωση φυσικής μνήμης - ΙΕΡΑΡΧΙΑ

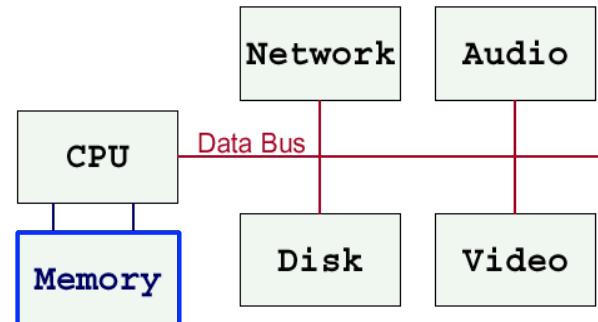


Indicated are approximate numbers of clock cycles to access the various elements of the memory hierarchy



Φυσική Μνήμη (physical memory)

- Χώρος αποθήκευσης μεταβλητών, δεδομένων, κώδικα, κλπ.



Ποιά από τις ακόλουθες σειρές αντικατοπρίζει καλύτερα την οργάνωση της ιεραρχίας της μνήμης ενός Η/Υ;

- Καταχωρητές, Σκληρός Δίσκος, RAM, Κεντρική Μνήμη
- Καταχωρητές, RAM, Σκληρός Δίσκος
- Καταχωρητές, Λανθάνουσα Μνήμη, RAM, Δίσκος
- Λανθάνουσα Μνήμη, Καταχωρητές, RAM, Δίσκος

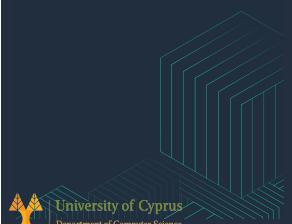
Διαχείριση Μνήμης

- Δέσμευση μνήμης (memory allocation), αποδέσμευση μνήμης
 - Τι σημαίνει;
- Ο καλός προγραμματισμός επιβάλλει την αποδοτική χρήση της μνήμης του Η/Υ.
- Είναι σημαντικό να καταλαβαίνουμε τις διαδικασίες διαχείρισης μνήμης:
 - Για την κατανόηση του τρόπου λειτουργίας των προγραμμάτων
 - Για την αποδοτική δέσμευση δομών δεδομένων μη προκαθορισμένου μεγέθους.
 - Για την αποφυγή «διαρροών μνήμης» (memory leaks).
 - Για την επίδοση του εκτελούμενου προγράμματος (run-time performance).

M. Δικαιάκος, ΕΠΛ133

9

ΤΙ ΕΙΝΑΙ Η ΥΠΟΛΟΓΙΣΤΙΚΗ ΔΙΕΡΓΑΣΙΑ (COMPUTATIONAL PROCESS)



M. D. Dikaiakos

Ενότητα 2: Διαχείριση Μνήμης και Σχεδιασμός Κλάσεων

Διεργασίες και Εικονική Μνήμη



Αφαίρεση στις Γ/Π



- Οι υπολογιστικές διεργασίες (computational processes) είναι αφαιρετικές οντότητες/υπολογιστικές δομές (abstract beings) που «υπάρχουν» και «εξελίσσονται» στους Η/Υ.
- Κατά την εξέλιξη τους, οι διεργασίες επεξεργάζονται άλλες αφαιρετικές οντότητες/δομές, που ονομάζονται δεδομένα.
- Η εξέλιξη μιας διεργασίας κατευθύνεται από μια σειρά κανόνων που κωδικοποιούνται στο πρόγραμμα που τρέχει (εντολές προγράμματος).
- Οι προγραμματιστές δημιουργούν προγράμματα για να κατευθύνουν τις υπολογιστικές διεργασίες.

Structure and Interpretation of Computer Programs, Abelson and Gerald Jay Sussman with Julie Sussman, MIT Press, 1996 1Λ133

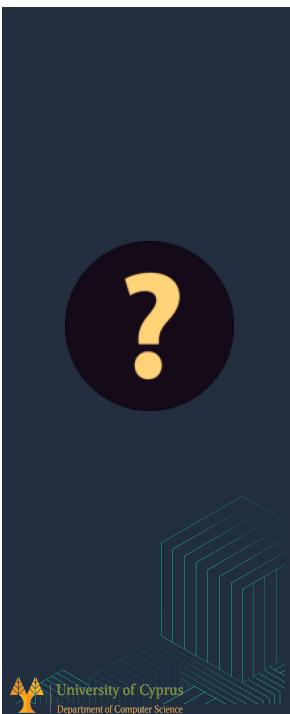
Τι είναι η διεργασία (process);

- Ένα πρόγραμμα που τρέχει
- Υπορουτίνα - συνάρτηση
- Ιθαγενής κώδικας ενός προγράμματος
- Ένα πρόγραμμα

00:30

13

ΕΠΛ133



ΠΩΣ ΒΛΕΠΟΥΜΕ ΠΟΙΕΣ
ΔΙΕΡΓΑΣΙΕΣ ΤΡΕΧΟΥΝ ΣΤΟΝ
ΥΠΟΛΟΓΙΣΤΗ ΜΑΣ;

M. D. Dikaiakos

```
mdd -- top - 80x32
Processes: 554 total, 4 running, 550 sleeping, 2013 threads 22:42:20
Load Avg: 2.22, 2.41, 2.64 CPU usage: 13.67% user, 11.55% sys, 74.76% idle
SharedLibs: 335M resident, 73M data, 28M linkedit.
MemRegions: 81373 total, 1883M resident, 99M private, 1434M shared.
PhysMem: 8127M used (2269M wired, 977M compressor), 64M unused.
VM: 18T vsize, 3570M framework vsize, 54988665(128) swapins, 59809442(0) swapout
Networks: packets: 25175457/20G in, 6369716/1458M out.
Disks: 39506419/715G read, 14320782/435G written.

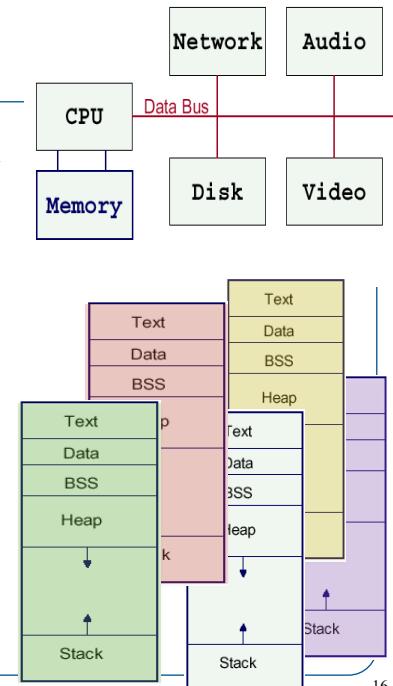
PID COMMAND %CPU TIME #TH #WQ #PORT MEM PURG CMPRS PGRP
534 WindowServer 29.7 04:23:26 13 6 2469+ 902M- 14M- 276M- 534
0 kernel_task 16.5 08:19:12 214/4 0 0 339M+ 0B 0B 0B 0
35831 com.apple.We 6.7 00:43.46 28/1 8 343+ 104M 0B 53M 35831
36050 top 5.2 00:01.45 1/1 0 29+ 4440K 0B 0B 36050
35562 Keynote 4.3 04:56.75 16 9 1315+ 1927M+ 653M- 529M- 35562
167 tccd 2.7 07:50.19 4 3 66+ 2756K+ 28K 896K 167
529 loginwindow 2.4 07:03.59 5 3 723+ 51M+ 0B 23M 529
35905 com.apple.We 2.1 00:19.74 10 4 96 220M+ 27M 13M 35905
137 launchservices 1.9 07:04.46 6 5 560+ 5924K+ 0B 908K 137
172 runningboard 1.7 16:18.01 7 6 666+ 5808K+ 0B 756K 172
16030 trustd 1.5 06:09.34 3 2 115 6272K 516K 2024K 16030
5191 WindowManage 1.2 02:13.19 5 2 274+ 11M 0B 6516K- 5191
1 launchd 1.2 16:28.51 5 3 3371+ 26M+ 0B 11M- 1
5346 Finder 1.2 10:11.44 7 5 959+ 146M+ 0B 94M- 5346
36040 Terminal 1.0 00:01.49 8 3 248 56M 544K 0B 36040
5277 knowledge-ag 0.7 12:39.42 4 3 285+ 10M+ 892K 4268K- 5277
159 notifyd 0.5 03:42.33 2 1 838+ 2196K 0B 308K 159
95 logd 0.5 16:02.76 4 3 1873+ 13M 0B 12M- 95
528 cfprefsd 0.5 05:24.85 3 2 583+ 2072K 176K 284K 528
35809 Safari 0.4 00:57.55 8 2 964 160M 0B 117M 35809
396 mds_stores 0.4 21:38.13 5 3 107+ 42M+ 0B 28M 396
124 mds 0.4 24:17.53 7 4 715+ 26M- 0B 16M 124
```

15

Μ. Δικαιάκος, ΕΠΛ133

Εικονική μνήμη

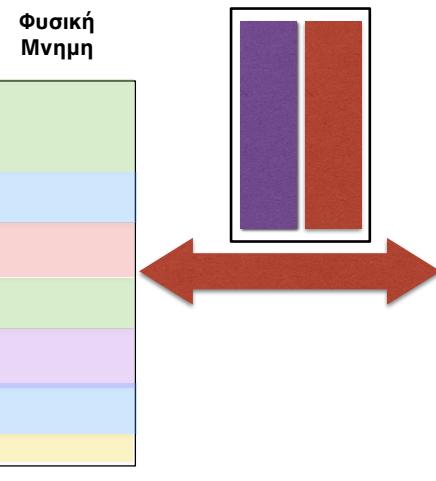
- Κατά την ενεργοποίηση μιας διεργασίας, το Λειτουργικό Σύστημα δημιουργεί έναν **εικονικό χώρο μνήμης** (virtual memory space) και έναν **εικονικό χώρο διευθύνσεων** (virtual address space) για την αποκλειστική χρήση της διεργασίας.
- Η διεργασία «βλέπει»/χρησιμοποιεί τις θέσεις μνήμης του εικονικού χώρου.
- Οι εικονικοί χώροι διευθύνσεων των διεργασιών που συν-τρέχουν στον Η/Υ, συνυπάρχουν τοποθετημένοι στην φυσική μνήμη του Η/Υ.
- **Χωρούν όλες οι διεργασίες στη Φυσική Μνήμη;**



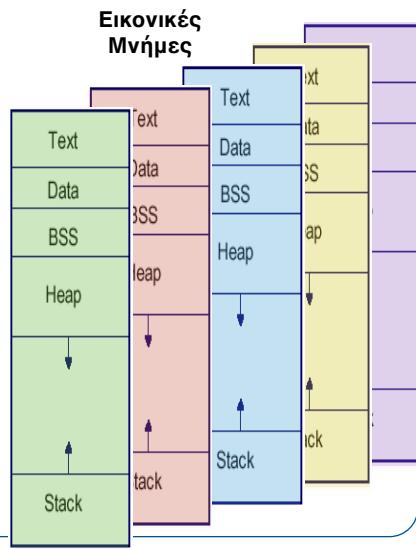
Μ. Δικαιάκος, ΕΠΛ133

16

Εικονική και φυσική μνήμη



M. Δικαιάκος, ΕΠΛ133



Ποια από τις ακόλουθες προτάσεις είναι σωστή;

00:30

01:00

- Η φυσική μνήμη είναι μεγαλύτερη από το σύνολο της εικονικής μνήμης όλων των διεργασιών που τρέχουν σε έναν Η/Υ
- Η φυσική μνήμη είναι μικρότερη από το άθροισμα των εικονικών μνημών των διεργασιών που τρέχουν σε έναν Η/Υ
- Ο χώρος διευθύνσεων μιας διεργασίας ανήκει στην εικονική μνήμη
- Η φυσική μνήμη πρέπει να είναι ίση με το άθροισμα των εικονικών μνημών των διεργασιών που τρέχουν σε έναν Η/Υ

ΕΠΛ133

Ενότητα 2: Διαχείριση Μνήμης και Σχεδιασμός Κλάσεων

Οργάνωση Εικονικής Μνήμης στο UNIX

ΠΩΣ ΕΙΝΑΙ ΟΡΓΑΝΩΜΕΝΗ Η
ΕΙΚΟΝΙΚΗ ΜΝΗΜΗ ΜΙΑΣ
ΔΙΕΡΓΑΣΙΑΣ;

Σχεδιάγραμμα της μνήμης μιας διεργασίας

- Text:** δυαδικός κώδικας (ιθαγενής / τελικός) του προγράμματος της διεργασίας.
- Data:** σταθερές προγράμματος που αρχικοποιούν γενικές και στατικές μεταβλητές κατά την έναρξη εκτέλεσης του προγράμματος.
- BSS (Block Started by Symbol (BSS):** γενικές και στατικές μεταβλητές (οι περισσότεροι συμβολομεταφραστές-assemblers στον κώδικα που παράγουν, κατασκευάζουν ένα τμήμα μνήμης BSS το οποίο συγκεντρώνει τον διαθέσιμο, δεσμευμένο μη αρχικοποιημένο χώρο).
- Heap:** δυναμική μνήμη.
- Stack:** στοίβα, τοπικές μεταβλητές.



21

Μ. Δικαιάκος, ΕΠΛ133

Σχεδιάγραμμα Μνήμης διεργασίας

- Στην Γ/Π C:

```
int iSize; ← BSS
```

```
char *f(void) {
```

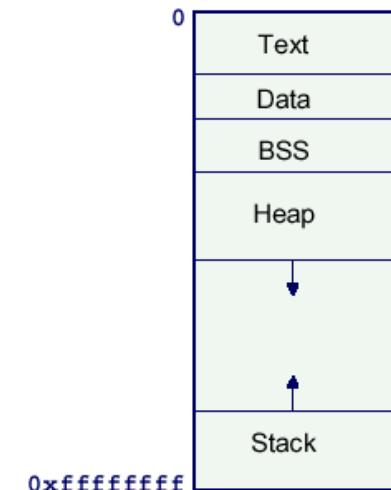
```
    char *p; ← stack
```

```
    iSize = 8; ← data
```

```
    p = malloc(iSize); ← heap
```

```
    return p;
```

```
}
```

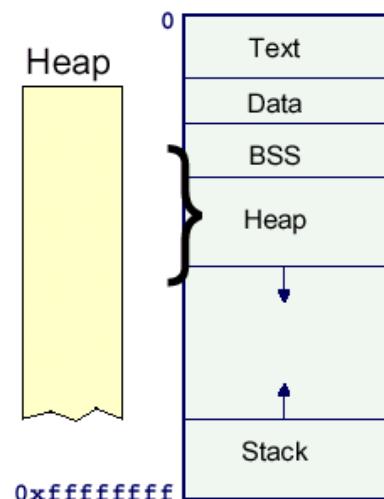


22

Μ. Δικαιάκος, ΕΠΛ133

Δέσμευση Μνήμης (memory allocation)

- Τι σημαίνει δέσμευση μνήμης;
- Πότε γίνεται η δέσμευση μνήμης;
 - Οι γενικές και οι στατικές μεταβλητές κατά την αρχικοποίηση του προγράμματος.
 - Οι τοπικές μεταβλητές κατά την κλήση μιας υπορουτίνας/μεθόδου.
 - Η δυναμική μνήμη με τη χρήση της `malloc` (στη C) και με τη δημιουργία νέων αντικειμένων με `new` στη JAVA.

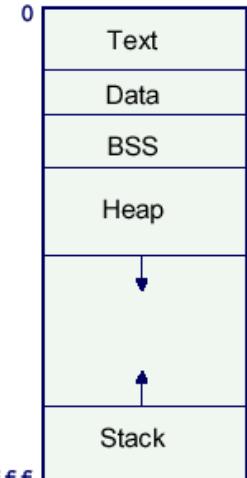


23

Μ. Δικαιάκος, ΕΠΛ133

Αποδέσμευση Μνήμης

- Γενικές και στατικές μεταβλητές με τη λήξη ενός προγράμματος.
- Τοπικές μεταβλητές με την επιστροφή από μια υπορουτίνα/μέθοδο.
- Δυναμική μνήμη με χρήση της `free` (στη C) και από τον αποκομιστή σκυβάλων (`garbage collector`) στη JAVA.
- Όλη η μνήμη αποδεσμεύεται με τη λήξη του προγράμματος.



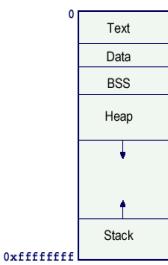
24

Μ. Δικαιάκος, ΕΠΛ133

Αποθηκευτικό Χώροι

▪ Καταχωρητές (Registers):

- Μέσα στον επεξεργαστή (CPU)
- Γρήγορη πρόσβαση
- Μικρός αριθμός
- Διαφανείς προς τον προγραμματιστή



▪ Στοίβα (Stack):

- Βρίσκεται στον εικονικό χώρο διευθύνσεων, στην κεντρική μνήμη (RAM)
- Απευθείας πρόσβαση από τον επεξεργαστή μέσω του **Δείκτη Στοίβας (Stack Pointer)**
- Χρησιμοποιείται για δεδομένα για τα οποία ο μεταφραστής μπορεί να γνωρίζει ακριβές μέγεθος και χρόνο ζωής (ώστε να διαχειρίζεται κατάλληλα τον Δ/Σ).
- Πολύ γρήγορη και αποδοτική πρόσβαση.
- Αποθηκεύει **εγγραφήματα δραστηριοποίησης** (activation records ή stack frames).
- Περιορισμένης λειτουργικότητας, γι' αυτό και χρησιμοποιείται κυρίως για την αποθήκευση ορισμένων μόνο δεδομένων και κυρίως τοπικών μεταβλητών μέσα στα εγγραφήματα δραστηριοποίησης.

Η αποθήκευση των δεδομένων

▪ Στατική Αποθήκευση (static storage)

- Βρίσκεται στον εικονικό χώρο διευθύνσεων, στην κεντρική μνήμη (RAM)
- Η στατική αποθήκευση υποδηλώνει την αποθήκευση δεδομένων σε συγκεκριμένη-αμετάβλητη θέση (fixed location).
- Η αποθήκευση στατικών δεδομένων γίνεται στην κεντρική μνήμη RAM.
- Περιέχει δεδομένα τα οποία είναι διαθέσιμα για όλη την διάρκεια εκτέλεσης του προγράμματος.

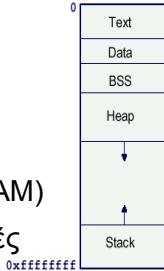
▪ Σταθερή Μνήμη (constant storage)

- Βρίσκεται στον εικονικό χώρο διευθύνσεων, στην κεντρική μνήμη (RAM)
- Αποθηκεύει τις σταθερές τιμές που χρησιμοποιούνται στον κώδικα.

Αποθηκευτικό Χώροι

▪ Σωρός (heap):

- Τοποθετείται στον εικονικό χώρο διευθύνσεων μνήμης (RAM)
- Μνήμη Γενικής χρήσης, όπου «διαβιούν» όλες οι δυναμικές δομές δεδομένων
- Πιο ευέλικτος από την στοίβα: ο μεταγλωττιστής δεν χρειάζεται να γνωρίζει εκ των προτέρων πόση μνήμη να δεσμεύσει στον σωρό για την αποθήκευση δομών δεδομένων, ή τον χρόνο ζωής των καταχωρούμενων δομών.
- Όποτε χρειαστεί να δημιουργηθεί μια δομή δεδομένων, πρέπει να να κληθεί η κατάλληλη μέθοδος για τη δέσμευση χώρου
- Πιό αργή η καταχώρηση και πρόσβαση απ' ότι στην στοίβα.



Αποθήκευση εκτός Κεντρικής Μνήμης

- Αφορά σε δεδομένα τα οποία «διαβιούν» τελείως εκτός του προγράμματος, ενώ το πρόγραμμα δεν τρέχει.

- Εκτός ελέγχου από το πρόγραμμα.

- Στη Java υπάρχουν δύο βασικά παραδείγματα:

- **Streamed objects:** αντικείμενα που μετατρέπονται σε ροή χαρακτήρων (byte stream) συνήθως για να σταλούν σε μιάλλη μηχανή.

- **Persistent objects:** αντικείμενα που τοποθετούνται στον δίσκο ώστε να διατηρήσουν την κατάσταση τους όταν το πρόγραμμα τελειώσει.

- Βασικό χαρακτηριστικό αυτών των αντικειμένων είναι ότι μπορούν να «μετατραπούν» σε κανονικά αντικείμενα στη RAM, μόλις αυτό χρειασθεί.

Περίγραμμα



- Εισαγωγή στη Διαχείριση Μνήμης
- Διαχείριση Μνήμης Διεργασιών
- **Διαχείριση Μνήμης Java κ. JVM**
- Αρχικοποιήσεις και Κατασκευή Αντικειμένων

M. D. Dikaiakos

Οργάνωση Μνήμης JVM (Run-time data areas)

- «Περιοχές δεδομένων» (data areas):
 - Της εικονικής μηχανής του JVM: δημιουργία κατά την εκκίνηση του JVM και καταστροφή κατά την έξοδο του
 - Του κάθε νήματος (thread) χωριστά: δημιουργία κατά τη δημιουργία του νήματος και καταστροφή με τη λήξη του.
- Καταχωρητής Μετρητή Προγράμματος (Program counter)
- Στοίβα του JVM (stack)
- Σωρός του JVM (heap)
- Περιοχή Μεθόδων (method area)
- Συλλογή Σταθερών Χρόνου Εκτέλεσης (Run-Time Constant Pool)
- Στοίβες Ιθαγενών Μεθόδων (Native Method Stacks)

Ενότητα 2: **Διαχείριση Μνήμης** και **Σχεδιασμός Κλάσεων**

Διαχείριση Μνήμης Java και JVM



Μετρητής Προγράμματος (Program Counter)

- Κάθε νήμα του Java Virtual Machine έχει τον δικό του καταχωρητή «μετρητή προγράμματος» - pc (program counter) register.
- Σε κάθε στιγμή, κάθε νήμα JVM εκτελεί τον κώδικα μίας μεθόδου (τρέχουσα μέθοδος) και ο μετρητής προγράμματος περιέχει την διεύθυνση της εκτελούμενης εντολής JVM.
 - Αν η εκτελούμενη μέθοδος είναι ιθαγενής (native) η τιμή του pc είναι ακαθόριστη.

Στοίβες JVM (JVM Stacks)

- Κάθε νήμα (thread) του JVM διαθέτει αποκλειστικά δική του στοίβα (JVM stack), η οποία δεσμεύεται κατά την ενεργοποίηση του νήματος.
- Πρόσβαση: μόνο από την κορυφή της στοίβας, με τη βοήθεια του δείκτη στοίβας (stack pointer)
- Λειτουργίες: push (εισαγωγή), pop (εξαγωγή)
- Περιεχόμενο στοίβας: εγγραφήματα δραστηριοποίησης (activation records) - πλαίσια στοίβας (stack frames):
 - τοπικές μεταβλητές, ορίσματα
 - ενδιάμεσα αποτελέσματα
 - πληροφορίες χρήσιμες για την κλήση μεθόδων και επιστροφή τιμών

Σωρός JVM (Heap)

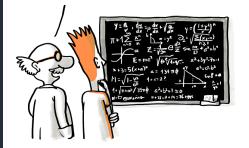
- Κάθε JVM διαθέτει ένα σωρό (heap) ο οποίος διαμοιράζεται μεταξύ των εκτελούμενων νημάτων
- Δημιουργείται κατά την εκκίνηση του JVM.
- Μπορεί να έχει σταθερό και συγκεκριμένο μέγεθος ή μεταβαλλόμενο (εξαρτάται από την υλοποίηση).
- Δεσμεύει χώρο για και αποθηκεύει όλα τα αντικείμενα: στιγμιότυπα των κλάσεων και των πινάκων.
- Δεν γίνεται ρητή αποδέσμευση χώρου - η απελευθέρωση χώρου γίνεται αυτόματα από τον αποκομιστή σκυβάλων (garbage collector).

Περιοχή Μεθόδων (method area)

- Ανάλογη του τμήματος "text" στο UNIX όπου αποθηκεύεται ο μεταγλωτισμένος κώδικας των συμβατικών Γ.Π. ή του τμήματος "text" της εικονικής μνήμης της διεργασίας στο Unix.
- Δημιουργείται κατά την έναρξη της εκτέλεσης του JVM.
- Αποτελεί μέρος του σωρού - ανάλογα με την υλοποίηση, μπορεί να έχει προκαθορισμένο ή ρυθμιζόμενο μέγεθος, να μην υπόκειται σε αποκομιδή σκυβάλων κλπ
- Αποθηκεύει δομές σχετικές με τις κλάσεις:
 - Κώδικας μεθόδων και κατασκευαστών
 - Συλλογή σταθερών χρόνου εκτέλεσης (run-time constant pool)

Αποθήκευση Αντικειμένων Java

- Τα δεδομένα των προγραμμάτων Java οργανώνονται και αποθηκεύονται κατά κύριο λόγο σαν αντικείμενα.
- Τα αντικείμενα αποθηκεύονται στο σωρό (heap).
- Για ένα αντικείμενο χρειαζόμαστε χειριστήριο (handle) μέσω του οποίου αποκτούμε πρόσβαση στο αντικείμενο και το διαχειρίζομαστε (τις μεθόδους του, τα δεδομένα του).
- Κάθε αντικείμενο διατηρεί την δική του μνήμη για τα δικά του πεδία δεδομένων. Οι θέσεις μνήμης που «κατέχει» ένα αντικείμενο δεν μπορούν να ανήκουν παράλληλα και σε άλλα αντικείμενα.



```
class Human {
    private String name;
    Human(String name) {
        this.name = name;
    }
}
```

```
class Car {
    private Human owner;
    private int vin;

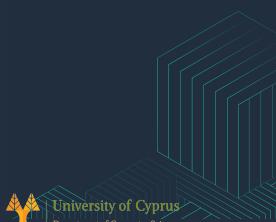
    Car(Human owner, int vin) {
        this.owner = owner;
        this.vin = vin;
    }
}
```

```
Human ann = new Human("Ann");
Car c1 = new Car(ann, 8);
Car c2 = new Car(ann, 10);
```

ΤΙ ΕΙΝΑΙ ΤΟ

ΠΕΔΙΟ ΕΜΒΕΛΕΙΑΣ

(SCOPE);



Αρχέγονοι Τύποι-primitive types

- Η δημιουργία αντικειμένων έχει κόστος, λόγω της αποθήκευσης στον σωρό και ό,τι αυτή συνεπάγεται (εντολές μηχανής για δέσμευση μνήμης, χρόνος εκτέλεσης).
- Για απλούς-αρχέγονους τύπους δεδομένων, η JAVA ξεφεύγει από τον μοντέλο των αντικειμένων και ακολουθεί την προσέγγιση της C και της C++:
 - Με την δήλωση μιας **τοπικής μεταβλητής** που έχει αρχέγονο τύπο, δεσμεύεται αυτόματα χώρος και η μεταβλητή αποθηκεύεται στο **εγγράφημα δραστηριοποίησης** της περικλείουσας μεθόδου της, στη **στοίβα (stack)** του νήματος που τρέχει.
 - Τι γίνεται για **πεδία δεδομένων** αρχέγονων τύπων;
- Ο ακριβής χώρος που κρατιέται για μεταβλητές αρχέγονων τύπων καθορίζεται από τον μεταφραστή της JAVA.
- Υπάρχουν όμως και **περιβάλλουσες κλάσεις** (wrapper classes) για τους αρχέγονους τύπους:

```
char c = 'x';   Character C = new Character('x');
```

M. Δικαιάκος, ΕΠΛ133

38

Πεδίο ισχύος - Εμβέλεια (scoping)

- Πεδίο ισχύος-εμβέλειας τοπικών μεταβλητών (scoping). Μια μεταβλητή που ορίζεται μέσα σε κάποιο πεδίο, είναι υπαρκτή μέχρι το τέλος του πεδίου αυτού.

```
{
    int x = 12;
    {
        int q = 96;
    }
}
```

- Στη Java το ακόλουθο είναι συντακτικό σφάλμα:

```
{
    int x = 12;
    {
        int x = 96;
    }
}
```

M. Δικαιάκος, ΕΠΛ133

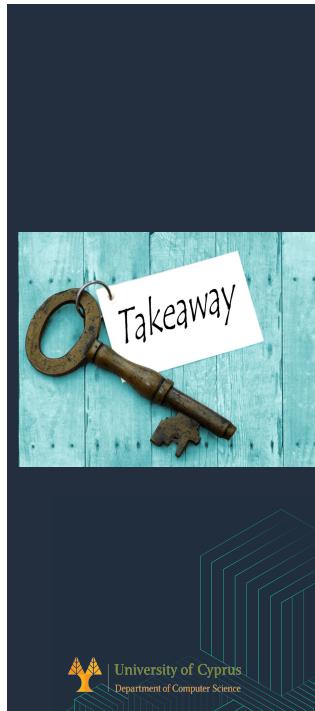
40

Διάρκεια ζωής αντικειμένων

- Ένα αντικείμενο που έχει δημιουργηθεί και τοποθετηθεί στο σωρό (με χρήση της `new`) διατηρείται στο σωρό ακόμη και μετά το τέλος του σχετικού πεδίου εμβέλειας (scope) στο οποίο δημιουργήθηκε.
- Ωστόσο, το αντίστοιχο χειριστήριο **χάνεται**:

```
{  
    String s = new String("a string");  
    /* end of scope */  
}
```
- Αποκομιστής σκυβάλων (garbage collector)

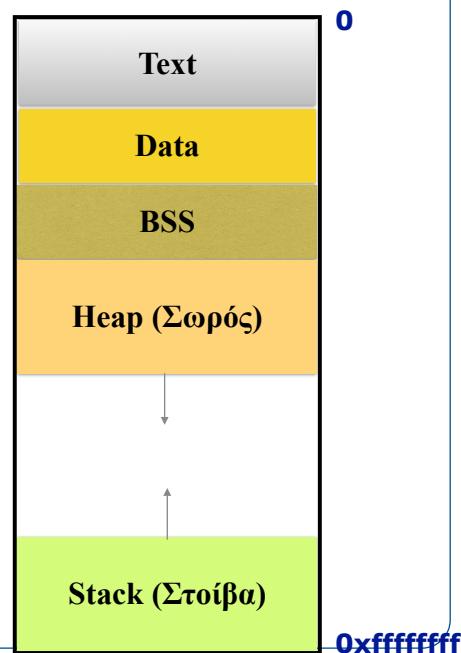
O προγραμματιστής δεν καταστρέφει αντικείμενα!



Κατά την εκτέλεση του JVM..



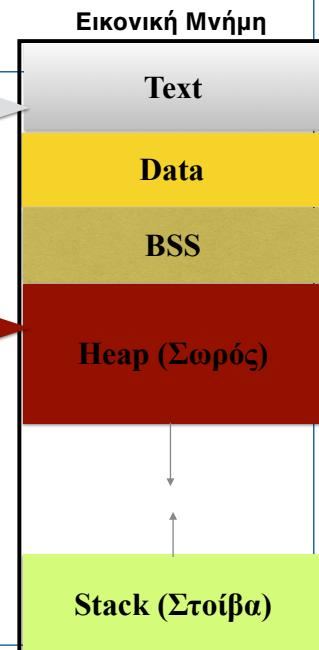
- Text: δυαδικός (ιθαγενής / τελικός) κώδικας προγράμματος της διεργασίας.
- Data: σταθερές προγράμματος.
- BSS (Block Started by Symbol (BSS)): γενικές και στατικές μεταβλητές (most modern assemblers produce a BSS section in their output object module containing all reserved uninitialized space).
- Heap: δυναμική μνήμη.
- Stack: στοίβα, τοπικές μεταβλητές.



Εκτέλεση Διεργασίας JVM

Τελικός κώδικας του προγράμματος JVM

- Δημιουργία/ενεργοποίηση διεργασίας του JVM.
- Ανάγνωση/εκτέλεση bytecodes από τον JVM.
 - Ο JVM δημιουργεί και διαχειρίζεται τον δικό του χώρο μνήμης
 - Ο χώρος μνήμης του JVM (**JVM run-time data areas**) υλοποιείται από το λογισμικό του JVM, και αξιοποιεί τον/βρίσκεται στο σωρό της διεργασίας του JVM



Περίγραμμα



- Οργάνωση και Διαχείριση Μνήμης
- Διεργασίες και Εικονική Μνήμη
- Διαχείριση Μνήμης Java κ. JVM
- **Αρχικοποιήσεις και Κατασκευή Αντικειμένων**
- Σκύβαλα και Αποκομιδή
- Στατικές Μεταβλητές και Μέθοδοι
- Περιβάλλουσες Κλάσεις
- Παράμετροι Κλάσεων
- Έλεγχος ισότητας αντικειμένων
- Αναλογιώτοι Περιορισμοί
- Παραβίαση ιδιωτικότητας και κατασκευαστές αντιγράφου

M. D. Dikaiakos

Initializers

- Στην Java, πριν την χρήση μιας μεταβλητής αρχέγονου τύπου, η μεταβλητή αυτή θα πρέπει να έχει αρχικοποιηθεί .
- Διαφορετικά ο μεταφραστής διαμαρτύρεται και βγάζει μήνυμα λάθους. Π.χ.:

```
int i;
i = i++; // illegal
```
- Για την αποφυγή του προβλήματος αυτού , χρησιμοποιούμε μια ειδική ανάθεση κατά τη δήλωση μιας μεταβλητής, η οποία λέγεται **αρχικοποιητής** (initializer), και με την οποία αρχικοποιούμε μια μεταβλητή αρχέγονου τύπου την στιγμή της δήλωσής της:

```
int i = 1;
```



Ενότητα 2: **Διαχείριση Μνήμης** και Σχεδιασμός Κλάσεων

Αρχικοποιήσεις και Κατασκευή Αντικειμένων



Αρχικοποίηση μεταβλητών στιγμιοτύπου

- Τα πεδία δεδομένων αντικειμένων (μεταβλητές στιγμιοτύπου - fields) **δεν** απαιτείται να αρχικοποιούνται από τον προγραμματιστή (*τι τιμές παίρνουν τότε ;*)
- Αν δεν κάνουμε εμείς ρητή αρχικοποίηση των μελών αρχέγονου τύπου μιας κλάσης, ο μεταγλωττιστής δίνει τις δικές του προκαθορισμένες τιμές, ως εξής:
 - Για boolean μεταβλητές: **false**
 - Για άλλους αρχέγονους τύπους το **0**.
 - Για χειριστήρια αντικειμένων: **null**
- Είναι καλή πρακτική να πραγματοποιούμε ρητά τις αρχικοποιήσεις μεταβλητών στιγμιοτύπου **μέσα στους κατασκευαστές**.



Παράδειγμα

```
class Measurement {  
    boolean t; char c; byte b; short s;  
    int i; long l; float f; double d;  
    void print() {  
        System.out.println("Data type Initial value\n" +  
            "boolean " + t + "\n" +  
            "char " + c + "\n" + "byte " + b + "\n" +  
            "short " + s + "\n" +  
            "int " + i + "\n" + "long " + l + "\n" +  
            "float " + f + "\n" + "double " + d);  
    }  
  
    public class InitialValues {  
        public static void main(String[] args){  
            new Measurement().print;  
        }  
    }  
}
```

M. Δικαιάκος, ΕΠΛ133

49

Αρχικοποιήσεις

```
class Measurement {  
    boolean t = false;  
    char c = 'z';  
    byte b = 9;  
    short s = 0;  
    int i = f(s);  
    long l = 12;  
    float f = 2.3;  
    double d = 0.9;  
    void print() { }  
} ... }
```

- Αρχικοποίηση μπορεί να γίνει με κλήση μεθόδου.
- Τα ορίσματα που δικαιούμαστε να περάσουμε στην μέθοδο αρχικοποίησης, πρέπει να έχουν ήδη αρχικοποιηθεί (αλλοιώς λαμβάνουμε exception).

M. Δικαιάκος, ΕΠΛ133

51

Εκροή Παραδείγματος

```
C:\users\epl233\eckel-code\c04>java InitialValues  
Data type Initial value  
boolean false  
char  
byte 0  
short 0  
int 0  
long 0  
float 0.0  
double 0.0
```

- Το **char** αρχικοποιείται σε null.
- Επίσης σε **null** αρχικοποιούνται και όποια χειριστήρια αντικειμένων περιέχονται ως στοιχεία σε αντικείμενα .

M. Δικαιάκος, ΕΠΛ133

50



Αρχικοποιήσεις και Κατασκευή Αντικειμένων

Κατασκευή αντικειμένων και Κατασκευαστές (Constructors)

Κατασκευή αντικειμένων

- Για τη δημιουργία νέου αντικειμένου χρησιμοποιούμε τον τελεστή `new`:

```
BankAcct b1 = new BankAcct(21338, 0.0);
String s = new String("asdf");
// String είναι ειδική περίπτωση τύπου
String s = "asdf";
```
- Το όνομα του κατασκευαστή (constructor) και η λίστα των (προαιρετικών) ορισμάτων του πρέπει υποχρεωτικά να ακολουθεί τον τελεστή `new`.
 - Αυτός είναι ο **μοναδικός έγκυρος τρόπος για κλήση κατασκευαστή**.
- Το νέο αντικείμενο μπορούμε να το αναθέσουμε σε ένα χειριστήριο (handle), ο τύπος του οποίου είναι ο ίδιος με την κλάση του αντικειμένου.

M. Δικαιάκος, ΕΠΑ133

53

Κλήση κατασκευαστών

- Ένας κατασκευαστής καλείται για τη δημιουργία αντικειμένου μιας κλάσης με χρήση της εντολής `new`

```
ClassName objectName = new ClassName(anyArgs);
```
- Εάν κληθεί ξανά ο κατασκευαστής (με χρήση `new`), δημιουργείται ένα νέο αντικείμενο.



M. Δικαιάκος, ΕΠΑ133

Copyright © 2017 Pearson Ltd. All rights reserved.

Κατασκευαστές (constructors)

- Ο **constructor** είναι μια ειδική κατηγορία μεθόδου που χρησιμοποιείται για την αρχικοποίηση ενός αντικειμένου κατά την στιγμή της δημιουργίας του.
- Οι κατασκευαστές δηλώνονται με το **ίδιο όνομα** με την κλάση στην οποία ανήκουν. Έτσι η ονομασία τους δεν χρειάζεται ιδιαίτερη διαχείριση, για αποφυγή τυχόν συγκρούσεων με άλλα ονόματα.
- Οι κατασκευαστές δεν επιστρέφουν τίποτε, χωρίς ωστόσο να δηλώνονται με τύπο επιστροφής `void`.
- Διευκολύνουν τον προγραμματισμό καθώς «ενοποιούν» ονοματολογικά την δήλωση και την αρχικοποίηση των κλάσεων και αντικειμένων.

M. Δικαιάκος, ΕΠΑ133

54

Εάν χρειαστεί να αλλάξετε τις τιμές των μεταβλητών στιγμιότυπου ενός υφιστάμενου αντικειμένου τι πρέπει να κάνετε;

00:30

- **Κάνετε κλήση της `new` με τον κατασκευαστή, στον οποίο περνάτε ως ορίσματα τις νέες τιμές.**
- **Κάνετε ανάθεση τις νέες τιμές απευθείας στις μεταβλητές στιγμιοτύπου.**
- **Κάνετε χρήση των κατάλληλων μεθόδων προσπέλασης, δίνοντας ως ορίσματα τις νέες τιμές.**
- **Κάνετε χρήση των κατάλληλων μεθόδων μεταλλαγής, δίνοντας ως ορίσματα τις νέες τιμές.**

ΕΠΑ133

56

Παράδειγμα constructor

```
class Rock {  
    Rock() { // This is the constructor  
        System.out.println("Creating Rock");  
    }  
}  
  
public class SimpleConstructor {  
    public static void main(String[] args) {  
        for(int i = 0; i < 10; i++)  
            new Rock();  
    }  
}
```

Κατασκευαστές και η παράμετρος `this`

- Η πρώτη ενέργεια που γίνεται από έναν κατασκευαστή είναι να δημιουργήσει αυτόματα ένα αντικείμενο, στο οποίο δεσμεύεται χώρος για τις μεταβλητές στιγμιότυπου που καθορίζει η κλάση του.
- Όπως οι (μη στατικές) μέθοδοι, έτσι και οι κατασκευαστές έχουν πρόσβαση στην παράμετρο `this`
- Στο σώμα ενός κατασκευαστή, η παράμετρος `this` υφίσταται και:
 - Αναφέρεται στο αντικείμενο που μόλις δημιουργήθηκε.
 - Μπορεί να χρησιμοποιηθεί ρητά για να αποκτήσουμε πρόβαση σε μεταβλητές στιγμιότυπου, αλλά πιο συχνά υπονοείται.

Λειτουργία κατασκευαστών

- Η πρώτη ενέργεια που γίνεται από έναν κατασκευαστή είναι να δημιουργήσει ένα αντικείμενο με μεταβλητές στιγμιότυπου (instance variables), όπως καθορίζονται από την κλάση του αντικειμένου
- Συνεπώς, είναι έγκυρη η κλήση μιας άλλης μεθόδου εντός του σώματος του κατασκευαστή, καθώς το νεοδημιουργημένο αντικείμενο είναι το αντικείμενο κλήσης για τη μέθοδο
 - Για παράδειγμα, μέθοδοι μεταλλαγής μπορεί να χρησιμοποιηθούν για τον ορισμό των τιμών των μεταβλητών στιγμιότυπου
 - Είναι επίσης δυνατό ένας κατασκευαστής να καλέσει έναν άλλο κατασκευαστή

Προκαθορισμένος κατασκευαστής (No-Argument Constructor)

- Εάν δεν συμπεριλάβετε κανέναν κατασκευαστή στην κλάση σας, η Java θα δημιουργήσει αυτόματα έναν δημόσιο, προεπιλεγμένο ([προκαθορισμένο - default](#)) κατασκευαστή, χωρίς παραμέτρους (*no-argument*)
- Ο προκαθορισμένος κατασκευαστής δεν δέχεται ορίσματα, δεν εκτελεί αρχικοποιήσεις, αλλά επιτρέπει τη δημιουργία του αντικειμένου
- Εάν συμπεριλάβετε έστω και έναν κατασκευαστή στην κλάση σας, η Java δεν θα παρέχει τον προκαθορισμένο κατασκευαστή, οπότε
 - φροντίστε να παρέχετε τον δικό σας κατασκευαστή χωρίς παραμέτρους, εφόσον τον χρειάζεστε.

Προκαθορισμένος κατασκευαστής (default constructor)

```
class Bird {  
    int i;  
}  
  
public class DefaultConstructorExample {  
    public static void main(String[] args) {  
        Bird nc = new Bird(); // default!  
        Bird nc = new Bird(1); // error!  
    }  
}
```

```
Bird() {  
    i = 0;  
}
```

Added by the compiler

M. Δικαιάκος, ΕΠΛ133

61

Κατασκευαστές με παραμέτρους

- Ένας constructor μπορεί να δεχεται παραμέτρους, οι οποίες καθορίζουν περαιτέρω το πώς θα αρχικοποιηθεί το αντίστοιχο αντικείμενο.

```
class Rock2 {  
    Rock2(int i) {  
        System.out.println("Creating Rock  
        number " + i);  
    }  
}  
public class SimpleConstructor2 {  
    public static void main(String[] args) {  
        for(int i = 0; i < 10; i++)  
            new Rock2(i);  
    }  
}
```

M. Δικαιάκος, ΕΠΛ133

62



Αρχικοποιήσεις και Κατασκευή Αντικειμένων

Overloading (Υπερφόρτωση)

Υπερφόρτωση Μεθόδων (overloading)

- Υπερφόρτωση μεθόδων προκύπτει στην Java όταν δύο ή περισσότερες μεθόδοι ή κατασκευαστές της ίδιας κλάσης, έχουν το ίδιο όνομα.
- Για να είναι έγκυρες, δηλώσεις μεθόδων με το ίδιο όνομα πρέπει να έχουν διαφορετικές **υπογραφές** (signatures).
- Η υπογραφή μιας μεθόδου αποτελείται από:
 - το όνομα της μεθόδου μαζί με
 - την λίστα των τυπικών παραμέτρων της (formal parameters).
- Διαφοροποιημένη υπογραφή δύο υπερφορτωμένων μεθόδων σημαίνει διαφορετικός αριθμός και/ή τύπος παραμέτρων.

Υπερφόρτωση

- Η υπερφόρτωση μεθόδων χρησιμοποιείται ευρύτατα στον Α/ΣΠ, **ιδιαίτερα** στους κατασκευαστές-constructors.
- Αν δύο μέθοδοι (κατασκευαστές) έχουν το ίδιο όνομα, τότε πως γνωρίζει η Java ποιά μέθοδο θέλουμε να καλέσουμε;
 - Κάθε υπερφορτωμένη μέθοδος πρέπει να δέχεται μια διαφορετική λίστα τύπων παραμέτρων.
 - Ακόμη και διαφορές στην σειρά των παραμέτρων είναι αρκετές για να ξεχωρίσουν δυό μεθόδους, αλλά αυτό δεν είναι «καλή» προγραμματιστική τακτική .

BAD PRACTICE

Παράδειγμα Υπερφόρτωσης

```
import java.util.*;  
class Tree {  
    int height;  
    Tree() {  
        prt("Planting a seedling"); height = 0;  
    }  
    Tree(int i) {  
        prt("Creating new Tree that is " + i +  
            " feet tall"); height = i;  
    }  
    void info() {  
        prt("Tree is " + height + " feet tall");  
    }  
    void info(String s) {  
        prt(s + ": Tree is " + height + " feet tall");  
    }  
    static void prt(String s) {  
        System.out.println(s); }  
}
```

Παράδειγμα Υπερφόρτωσης (συνέχεια)

```
public class Overloading {  
  
    public static void main(String[] args) {  
  
        for(int i = 0; i < 5; i++) {  
  
            Tree t = new Tree(i);  
  
            t.info();  
  
            t.info("overloaded method");  
  
        }  
  
        // Overloaded constructor:  
  
        new Tree();  
  
    }  
}
```

Υπερφόρτωση και Αρχέγονοι τύποι

- Αν ορίσουμε μια μέθοδο : **void f (double x) { /* */}** και την καλέσουμε σαν **f(5)**, η παράμετρος «5» θα προαχθεί αυτόματα (promotion) σε 5.0.
- Τι θα συμβεί αν η **f()** είναι υπερφορτωμένη ώστε να δέχεται και ακέραιη παράμετρο;
 - Θα εκτελεσθεί η έκδοση της **f()** που δέχεται ακέραιες παραμέτρους.
- Γενικότερα, αν η τιμή που περνάμε σε μια μέθοδο έχει τύπο μικρότερο από τον τύπο της δηλωμένης παραμέτρου, τότε η τιμή προάγεται στον τύπο της παραμέτρου.
- Αν υπάρχουν πολλοί τέτοιοι δυνατοί τύποι (πολλαπλών «υπερφορτώσεων»), τότε η προαγωγή γίνεται στον αμέσως μεγαλύτερο τύπο.

Υπερφόρτωση και Αρχέγονοι τύποι

- Εξαίρεση έχουμε στην περίπτωση που ο τύπος της δηλωμένης παραμέτρου είναι char κι εμείς περνάμε έναν ακέραιο, ο οποίος δεν αντιστοιχεί σε τιμή χαρακτήρα char, τότε η τιμή αυτή προάγεται σε ακέραιο (και όχι, π.χ. σε short).

byte → short → int → long → float → double
char —————↑

M. Δικαιάκος, ΕΠΛ133

69

Παράδειγμα

```
public class Foo {  
    void f1(double x) {  
        System.out.println("double f1 -->" + x); }  
    void f1(int x) {  
        System.out.println("int f1 -->" + x); }  
    public static void main(String[] args) {  
        Foo ff = new Foo();  
        ff.f1(5);  
    } }
```

M. Δικαιάκος, ΕΠΛ133

70

Κλήση κατασκευαστών από κατασκευαστές

- Η υπογραφή (signature) μιας μεθόδου περιλαμβάνει μόνο το όνομα και τους τύπους των τυπικών παραμέτρων της μεθόδου:
 - Δεν περιλαμβάνει τον επιστρεφόμενο τύπο (type returned)
 - Η Java δεν επιτρέπει μεθόδους με την ίδια υπογραφή και διαφορετικούς τύπους επιστροφής μέσα στην ίδια κλάση
- Ambiguous method invocations will produce an error in Java

BAD PRACTICE

Copyright © 2017 Pearson Ltd. All rights reserved.

M. Δικαιάκος, ΕΠΛ133

71

M. Δικαιάκος, ΕΠΛ133

72

Παράδειγμα

```
public class Foo {  
    void f1(double x) {  
        System.out.println("double f1 -->" + x); }  
    void f1(int x) {  
        System.out.println("int f1 -->" + x); }  
    public static void main(String[] args) {  
        Foo ff = new Foo();  
        ff.f1(5);  
    } }
```

M. Δικαιάκος, ΕΠΛ133

70

Κλήση κατασκευαστών από κατασκευαστές

- Μέσα σε έναν **constructor**, η δεσμευμένη λέξη **this** μπορεί να χρησιμοποιηθεί με σύνταξη κλήσης μεθόδου: **this()**;
- Σε αυτή την περίπτωση, η **this()**; προκαλεί ρητή κλήση μιας διαφορετικής εκδοχής του constructor, η οποία εκδοχή έχει δηλωθεί με υπερφόρτωση.
- Η **this()**; μπορεί να χρησιμοποιηθεί σαν κλήση **constructor**:
 - μέσα σε κάποιον constructor,
 - μόνο σαν **πρώτη εντολή του constructor** και
 - μόνο για **μια φορά**.

Παράδειγμα

```
// Calling constructors with "this"
public class Flower {
    private int petalCount = 0;
    private String s = new String("null");
    Flower(int petals) {
        petalCount = petals;
    }
    Flower(String ss) {
        s = ss;
    }
}
```

M. Δικαιόκος, ΕΠΛ133

73

Παράδειγμα (συνέχεια)

```
Flower(String s, int petals) {
    this(petals);
    this(s); // Δεν επιτρέπεται δεύτερη κλήση του this
    this.s = s;
}
Flower() {
    this("hi", 47);
}
```

M. Δικαιόκος, ΕΠΛ133

74

Παράδειγμα (συνέχεια)

```
void print() {
    this(11); // Δεν επιτρέπεται η κλήση του this εκτός constructor
    System.out.println("petalCount = " + petalCount +
        " s = " + s);
}

public static void main(String[] args) {
    Flower x = new Flower();
    x.print();
}
```

M. Δικαιόκος, ΕΠΛ133

75