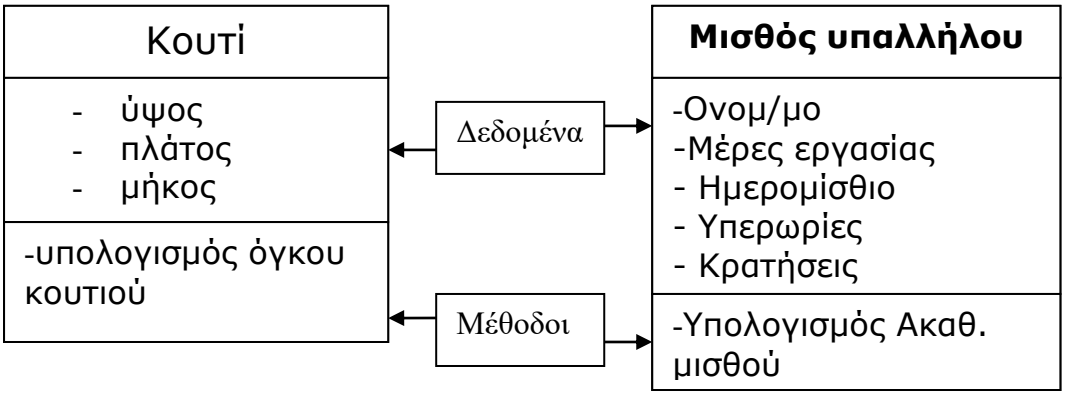
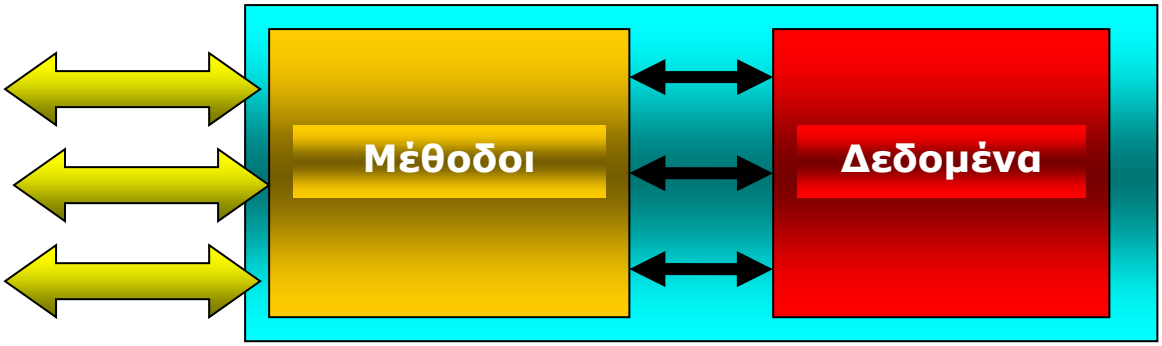
ΠΡΟΣΑΝΑΤΟΛΙΣΜΟΣ ΣΕ ΑΝΤΙΚΕΙΜΕΝΑ

Προσανατολισμός σε αντικείμενα είναι η προσέγγιση που χρησιμοποιεί συλλογές διακριτών ***αντικειμένων*** στην ανάπτυξη λογισμικού για την επίλυση προβλημάτων. **Αντικείμενο** είναι ένα δομικό συστατικό λογισμικού που έχει **κατάσταση, συμπεριφορά** και **ταυτότητα.** Η κατάσταση περιγράφει τις *στατικές ιδιότητες* του αντικειμένου (τιμές των μεταβλητών). Οι ιδιότητες αυτές είναι αποτέλεσμα της συμπεριφοράς του αντικειμένου, δηλαδή μιας *ενέργειας ή μετασχηματισμού* που το αντικείμενο *εκτελεί ή στην οποία υπόκειται* (ανταπόκριση σε κλήσεις από το περιβάλλον του). Μια συγκεκριμένη υλοποίηση μιας συμπεριφοράς λέγεται **μέθοδος.** Η ταυτότητα είναι η μοναδική *διάκριση του αντικειμένου* από τα ομοειδή του (καταστάσεις και συμπεριφορές που συσχετίζονται μαζί του).



Τα αντικείμενα επικοινωνούν μεταξύ τους με μηνύματα. Στα αντικείμενα γίνεται η **απόκρυψη των δεδομένων** *(data hiding)* από τον εξωτερικό κόσμο, ενώ η προσπέλαση της πληροφορίας γίνεται μέσω ενός **συνόλου διαδικασιών** *(object interface).*

Στατικό μέρος (private)	**➔**	**Δεδομένα (Μεταβλητές)**
Δυναμικό μέρος (public)	**➔**	**Διαδικασίες (Μέθοδοι)**



Εικ. 1. Προσπέλαση αντικειμένων

Ορισμοί

Κλάση *(class)*

Το σύνολο των αντικειμένων που έχουν την ίδια δομή και την ίδια συμπεριφορά, ονομάζεται κλάση ή διαφορετικά η κλάση περιγράφει μια οικογένεια αντικειμένων με ίδια χαρακτηριστικά και συμπεριφορά. Π.χ. βιβλίο, κατοικίδιο, κ.λ.π. Οι κλάσεις ενός πεδίου εφαρμογής (π.χ. μία εφαρμογή μισθοδοσίας) είναι οργανωμένες σε μια ή περισσότερες **ιεραρχίες κλάσεων *(class hierarchy).*** Στην ιεραρχία, μία κλάση μπορεί να έχει μια ή περισσότερες **υποκλάσεις *(subclasses)*** και μία τουλάχιστον **υπερκλάση *(superclass).***

Στιγμιότυπο *(instance)*

Κάθε αντικείμενο αποτελεί ένα μοναδικό στιγμιότυπο ή εκδοχή της κλάσης που ανήκει ή διαφορετικά είναι ένα συγκεκριμένο αντικείμενο, που παράγεται από την αντίστοιχη κλάση, στην οποία ανήκει. Π.χ. λεξικό, γάτα.

Πεδίο *(field ή attribute)*

Μεταβλητή που παριστάνει ένα χαρακτηριστικό γνώρισμα του αντικειμένου, π.χ. ύψος κουτιού, ημερομίσθιο υπαλλήλου.

Μέθοδος *(method)*

Συστατικό λογισμικού (διαδικασία ή συνάρτηση) που υλοποιεί μια συμπεριφορά ενός αντικειμένου.

Ενθυλάκωση ή Κελυφοποίηση ή απόκρυψη πληροφοριών *(encapsulation)*

Η απόκρυψη των λεπτομερειών υλοποίησης ενός αντικειμένου (ελεγχόμενη ορατότητα των πεδίων και μεθόδων του αντικειμένου).

Δημόσια (public) πεδία/μέθοδοι: αυτά που μπορούν να χρησιμοποιηθούν (κληθούν/ προσπελαστούν) από τα άλλα αντικείμενα.

Ιδιωτικά (private) πεδία/μέθοδοι: αυτά που χρησιμοποιούνται μόνο από το ίδιο το αντικείμενο.

ΣΧΕΣΕΙΣ ΜΕΤΑΞΥ ΚΛΑΣΕΩΝ

Κληρονομικότητα ή γενίκευση *(inheritance - generalization)*

Όταν μία κλάση αποδίδει σε μία άλλη κλάση τα χαρακτηριστικά της. Η πρώτη κλάση λέγεται κλάση-γονέας η δε δεύτερη κλάση-παιδί, και μπορεί να διαθέτει επιπλέον χαρακτηριστικά (πεδία και μεθόδους) από αυτά που κληρονομεί. Σε μία σχέση κληρονομικότητας με ένα μόνο γονέα έχουμε την **απλή κληρονομικότητα *(single inheritance),*** ενώ στην αντίθετη περίπτωση **πολλαπλή κληρονομικότητα *(multiple inheritance).*** Αν δούμε την κληρονομικότητα από την κλάση-παιδί προς την κλάση-γονέα, τότε παρατηρούμε ότι η κλάση-γονέας έχει λιγότερα χαρακτηριστικά από την κλάση-παιδί, δηλαδή αποτελεί **γενίκευσή *(generalization)*** της. Με την κληρονομικότητα έχουμε και τον ορισμό της **επαναχρησιμοποίησης** (δεδομένων και λειτουργιών).

Πολυμορφισμός *(polymorphism)*

Πολυμορφισμός είναι η δυνατότητα εκτέλεσης διαφορετικών λειτουργιών μιας μεθόδου που είναι κοινή σε διαφορετικές κλάσεις (συνήθως κλάση-γονέα και κλάση-παιδί), ανάλογα με το αντικείμενο της κλάσης που την καλεί. Είναι μία σημαντική ιδιότητα του αντικειμενοστρεφούς προγραμματισμού που χρησιμοποιείται ευρέως στην επεκτασιμότητα των εφαρμογών.

Συσχέτιση *(association)*

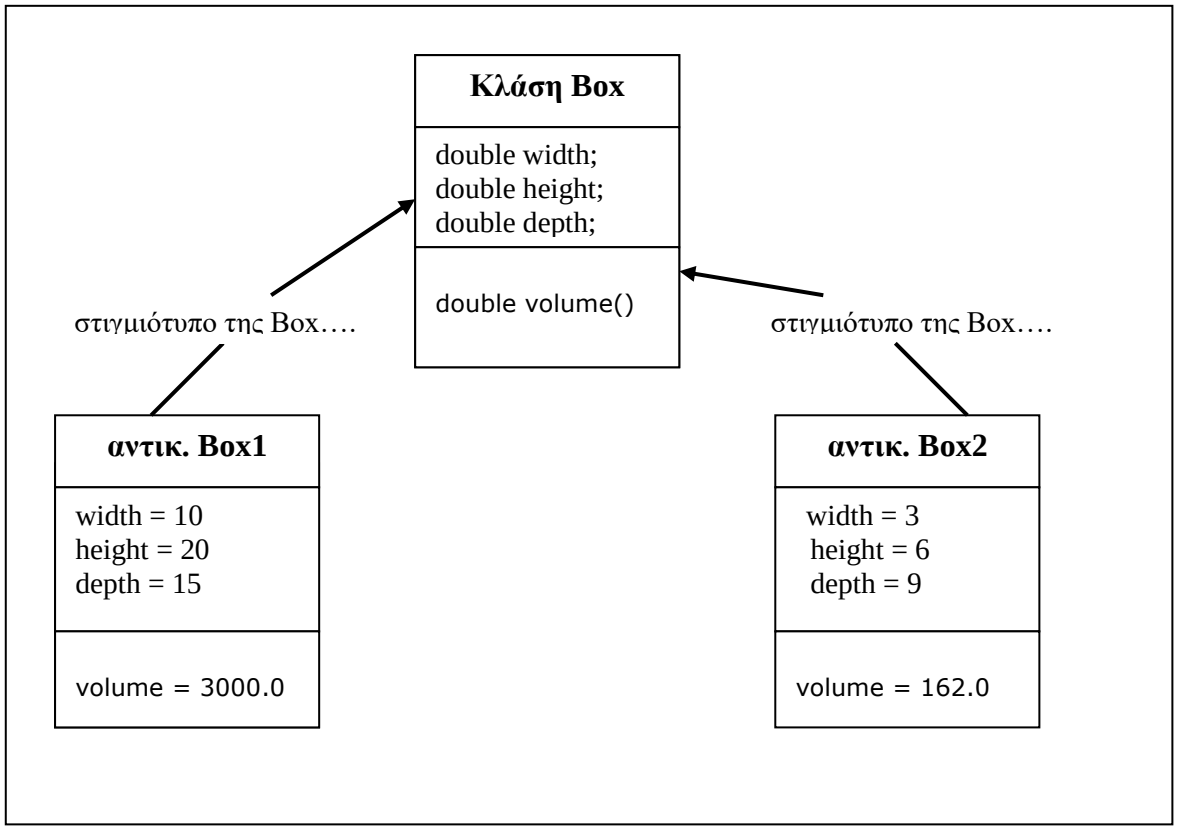
Μία εννοιολογική σύνδεση ή σχέση ανάμεσα σε δύο ή περισσότερες κλάσεις. Οι συσχετίσεις είναι ‘ένα προς ένα’, ‘ένα προς πολλά’, και ‘πολλά προς πολλά’. Για παράδειγμα η σχέση ανάμεσα στον φοιτητή και τα μαθήματα του εξαμήνου. Ο φοιτητής μπορεί να πάρει από 1 έως 6 μαθήματα (ένας προς πολλά).

Συναρμολόγηση *(aggregation)*

Η σχέση που εκφράζει την σύνθεση του συνόλου από απλούστερα μέρη. Απεικονίζει τη σχέση 'αποτελείται-από' *(consists-of)*, 'περιέχει' *(contains)*, 'έχει' *(has)*, 'είναι ένα άθροισμα από' (is an aggregation of). Για παράδειγμα το αυτοκίνητο έχει τιμόνι, μηχανή, κιβώτιο ταχυτήτων, ρόδες, κλπ.

Μια πρώτη εικόνα υλοποίησης κλάσεων, αντικειμένων και μεθόδων στον αντικειμενοστρεφή προγραμματισμό

1. Θα δημιουργήσουμε μία κλάση αντικειμένων boxes (κουτιά) που θα έχουν σαν πεδία το μήκος, ύψος και πλάτος του box και μία μέθοδο υπολογισμού του όγκου του.
2. Θα δημιουργήσουμε δύο αντικ. τύπου box. Το 1ο θα έχει τιμές 10, 20, 15 και το 2ο θα έχει αντίστοιχα τις τιμές 3, 6, 9.
3. Θα υλοποιήσουμε ένα μικρό πρόγραμμα Java που θα υπολογίζει και εμφανίζει τον όγκο των δύο κουτιών.



Το πρόγραμμα σε Java

Κλάση Αντικειμένων

```
class Box {

  double width;
  double height;
  double depth;

  // ο δομητής του Box.
  Box(double x, double y, double z) {
    System.out.println("Constructing Box");
    width = x;
    height = y;
    depth = z;
  }

  // υπολογισμός και επιστροφή του όγκου
   double volume() {
    return width * height * depth;
  }
}
```

Κλάση Box που ορίζει (ή αποτελείται από) αντικείμενα αυτού του τύπου. Τα αντικείμενα έχουν **πεδία** (χαρακτηριστικά) το μήκος, ύψος και πλάτος.

Ο **δομητής ή κατασκευαστής** των αντικειμένων, αρχικοποιεί τα αντι-Κείμενα.

Η **μέθοδος** υπολογισμού του όγκου του αντικειμένου – box.

Κλάση Προγράμματος

```
class BoxDemo {
  public static void main(String args[]) {
    // αρχικοποίηση αντικειμένων
   Box mybox1 = new Box(10, 20, 15);
   Box mybox2 = new Box(3, 6, 9);

    double vol;

    // λαμβάνεται ο όγκος του πρώτου box
    vol = mybox1.volume();
    System.out.println("Volume is " + vol);

    // λαμβάνεται ο όγκος του δευτέρου box
    vol = mybox2.volume();
    System.out.println("Volume is " + vol);
  }
}
```

Κλάση που **εκτελεί λειτουργίες** στα αντικείμενα τύπου – box

Δημιουργία δύο αντικειμένων

Ορισμός χώρου μνήμης για τη λήψη της τιμής του όγκου

Λήψη και **εμφάνιση** του όγκου του 1ου – box (3000.0)

Λήψη και **εμφάνιση** του όγκου του 2ου – box (162.0)

Κλάσεις, Αντικείμενα και Μέθοδοι

Κλάσεις

Η κλάση *(class)* είναι ένα **πρότυπο** *(template)* παραγωγής αντικειμένων. Τα αντικείμενα λέγονται **στιγμιότυπα** *(instances)* της κλάσης. Με τον ορισμό μιας κλάσης ορίζονται:

- τα **πεδία – μεταβλητές** *(variables)* που θα εμπεριέχει,
- ο/οι **δομητής/-ές** ή **κατασκευαστής/-ές** *(constructors)* καθώς και
- οι **μέθοδοι** *(methods),* δηλαδή τα κομμάτια του κώδικα που θα χειρίζονται αυτά τα δεδομένα.

```
class <όνομα κλάσης>
{
type instance – μεταβλητή-1
 :              :       :
type instance – μεταβλητή-n

<όνομα δομητή – κλάσης - 1> [(λίστα παραμέτρων)]
 {
      //σώμα δομητή – κατασκευαστή
 }
       :              :              :
<όνομα δομητή – κλάσης – n> [(λίστα παραμέτρων)]
 {
      //σώμα δομητή – κατασκευαστή
 }

type <όνομα-μεθόδου-1> [(λίστα παραμέτρων)]
 {
      //σώμα της μεθόδου
 }
      :              :              :
type <όνομα-μεθόδου-n> [(λίστα παραμέτρων)]
 {
      //σώμα της μεθόδου
 }
}
```

Πεδία – Μεταβλητές

Δομητές – Κατασκευαστές αντικειμένων

Κώδικας – Μέθοδοι

Οι μεταβλητές-πεδία και οι μέθοδοι που εμπεριέχει μια κλάση ονομάζονται **μέλη** της κλάσης *(class members)*. Μια κλάση μπορεί να περιέχει μόνο δεδομένα ή μόνο κώδικα, συνήθως όμως περιέχει και τα δύο. Ο ορισμός ξεκινά με τη δεσμευμένη λέξη **class** και το όνομα της κλάσης. Το όνομα της κλάσης ακολουθεί τους κανόνες σχηματισμού ονομάτων δηλαδή περιέχει γράμματα, αριθμούς (όχι σαν πρώτοι χαρακτήρες), και τα σύμβολα ($) και (_). Ακολουθούν οι δηλώσεις των πεδίων - μεταβλητών που ανάλογα με τον τρόπο δήλωσής των χαρακτηρίζονται είτε σαν **μεταβλητές αντικειμένου ή στιγμιότυπου** *(instance variables),* ή σαν **μεταβλητές κλάσης** *(class variables).* Οι μεταβλητές αντικειμένου ή στιγμιότυπου δηλώνονται μόνο με τον τύπο και το ονομά τους (χωρίς τιμές), γιατί κάθε αντικείμενο της κλάσης θα έχει τις δικές του τιμές. Οι μεταβλητές κλάσης δηλώνονται με τιμές και με τον χαρακτηρισμό **static,** που σημαίνει ότι όλα τα αντικείμενα της κλάσης θα έχουν τις ίδιες τιμές. Όταν η τιμή μιας μεταβλητής κλάσης αλλάξει, τότε η νέα τιμή θα ισχύει για

όλα τα αντικείμενα. Οι μεταβλητές των κλάσεων μπορεί να περιέχουν τιμές των βασικών τύπων της Java, ή και αναφορές σε άλλα αντικείμενα.

Μετά τις δηλώσεις των μεταβλητών ακολουθεί η δήλωση του/των **δομητή/-ών** – **κατασκευαστή/-ών των αντικειμένων** που είναι το κομμάτι του κώδικα το οποίο κατασκευάζει και αρχικοποιεί τα αντικείμενα (θα επανέλθουμε μετά την παρουσίαση των μεθόδων).

Μετά τη δήλωση του/των δομητών - κατασκευαστών ακολουθεί η δήλωση των μεθόδων, δηλαδή των τμημάτων του κώδικα που εκτελούν πράξεις και ενεργούν στα πεδία – μεταβλητές. Οι μέθοδοι έχουν ένα τύπο, ένα όνομα και παρενθέσεις μέσα στις οποίες περικλείουμε **παραμέτρους** για να λειτουργήσει η μέθοδος (αν δεν περιλαμβάνονται παράμετροι οι παρενθέσεις μένουν κενές αλλά δεν παραλείπονται). Ο τύπος, το όνομα της μεθόδου και οι παράμετροι αποτελούν την **υπογραφή** *(signature)* της μεθόδου. Δύο είναι οι τύποι των μεθόδων: οι **μέθοδοι κλάσης** και οι **μέθοδοι στιγμιότυπου.** Οι μέθοδοι κλάσης δηλώνονται με τη λέξη κλειδί **static** πριν από τον τύπο και αυτό σημαίνει ότι μπορούν να εκτελέσουν τις λειτουργίες τους χωρίς να έχει δημιουργηθεί αντικείμενο της κλάσης. Το αντίθετο συμβαίνει σε μια κλάση στιγμιότυπου. Δηλαδή για να εκτελεστεί ο κώδικας μιας κλάσης στιγμιότυπου θα πρέπει πρώτα να δημιουργηθεί ένα αντικείμενο της κλάσης. Μια μέθοδος μπορεί να **επιστρέφει** (αφού πρώτα υπολογίσει) μια **τιμή** (του ίδιου τύπου – της δήλωσης) χρησιμοποιώντας τη λέξη κλειδί **return.** Αν απλώς εκτελεί λειτουργίες και πράξεις χωρίς να επιστρέφει τιμή (αλλά τη ροή του προγράμματος) τότε συντάσσεται με τη λέξη κλειδί **void** πριν από τον τύπο.

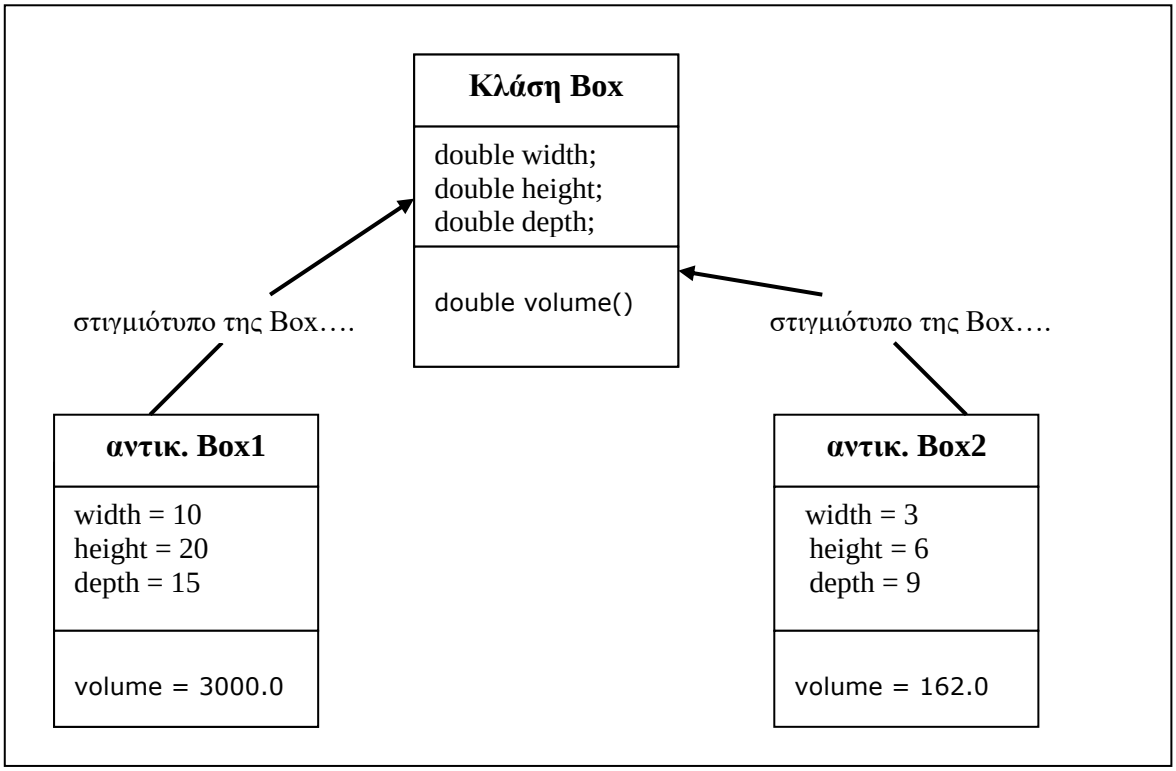
Η **μέθοδος κατασκευής αντικειμένων (δομητής – κατασκευαστής)** είναι τύπου public (προσπελάσιμη από όλες τις κλάσεις του προγράμματος), έχει το ίδιο όνομα με την αντίστοιχη κλάση και παρόμοια σύνταξη με μια μέθοδο. Μόνο που δεν επιστρέφει τιμή, ούτε όμως συντάσσεται με τη λέξη κλειδί void. Χρησιμοποιείται για να δώσει αρχικές τιμές στα πεδία – μεταβλητές των αντικειμένων (που καθορίζονται από την κλάση) ή για να εκτελέσει άλλες αναγκαίες διαδικασίες έναρξης για την δημιουργία των αντικειμένων. ***Όλες οι κλάσεις έχουν απ' αρχής (από κατασκευής) ένα δομητή – κατασκευαστή που δίνει μηδενικές τιμές στα πεδία – μεταβλητές των αντικειμένων.*** Ωστόσο όταν εμείς γράψουμε τον δικό μας δομητή – κατασκευαστή, τότε παύει να ισχύει ο αρχικός και ισχύει αυτός που γράψαμε. Η προσθήκη παραμέτρων στον δομητή – κατασκευαστή γίνεται όπως και στις μεθόδους (**<τύπος1> <όνομα1>, <τύπος2> <όνομα2>,** κλπ.>). Μπορούμε να γράψουμε πολλούς διαφορετικούς δομητές - κατασκευαστές για να δημιουργούμε αντικείμενα με διαφορετικές αρχικές τιμές.

Μη ξεχνάμε ότι τα προγράμματα της Java περιέχουν:

1) μία κλάση με τη μέθοδο **main(),** από όπου ξεκινά η εκτέλεση του προγράμματος και από όπου συνήθως δημιουργούνται αντικείμενα άλλων κλάσεων και καλούνται οι αντίστοιχοι μέθοδοι για να εκτελέσουν οι λειτουργίες και πράξεις με τα δεδομένα και
2) μία ή περισσότερες κλάσεις πρότυπα για τη δημιουργία αντικειμένων.

Παράδειγμα-1: Ένα απλό πρόγραμμα υλοποίησης κλάσεων, αντικειμένων και μεθόδων

4. Θα δημιουργήσουμε μία κλάση αντικειμένων boxes (κουτιά) που θα έχουν σαν πεδία το μήκος, ύψος και πλάτος του box και μία μέθοδο υπολογισμού του όγκου του.
5. Θα δημιουργήσουμε δύο αντικ. τύπου box. Το 1ο θα έχει τιμές 10, 20, 15 και το 2ο θα έχει αντίστοιχα τις τιμές 3, 6, 9.
6. Θα υλοποιήσουμε ένα μικρό πρόγραμμα Java που θα υπολογίζει και εμφανίζει τον όγκο των δύο κουτιών.



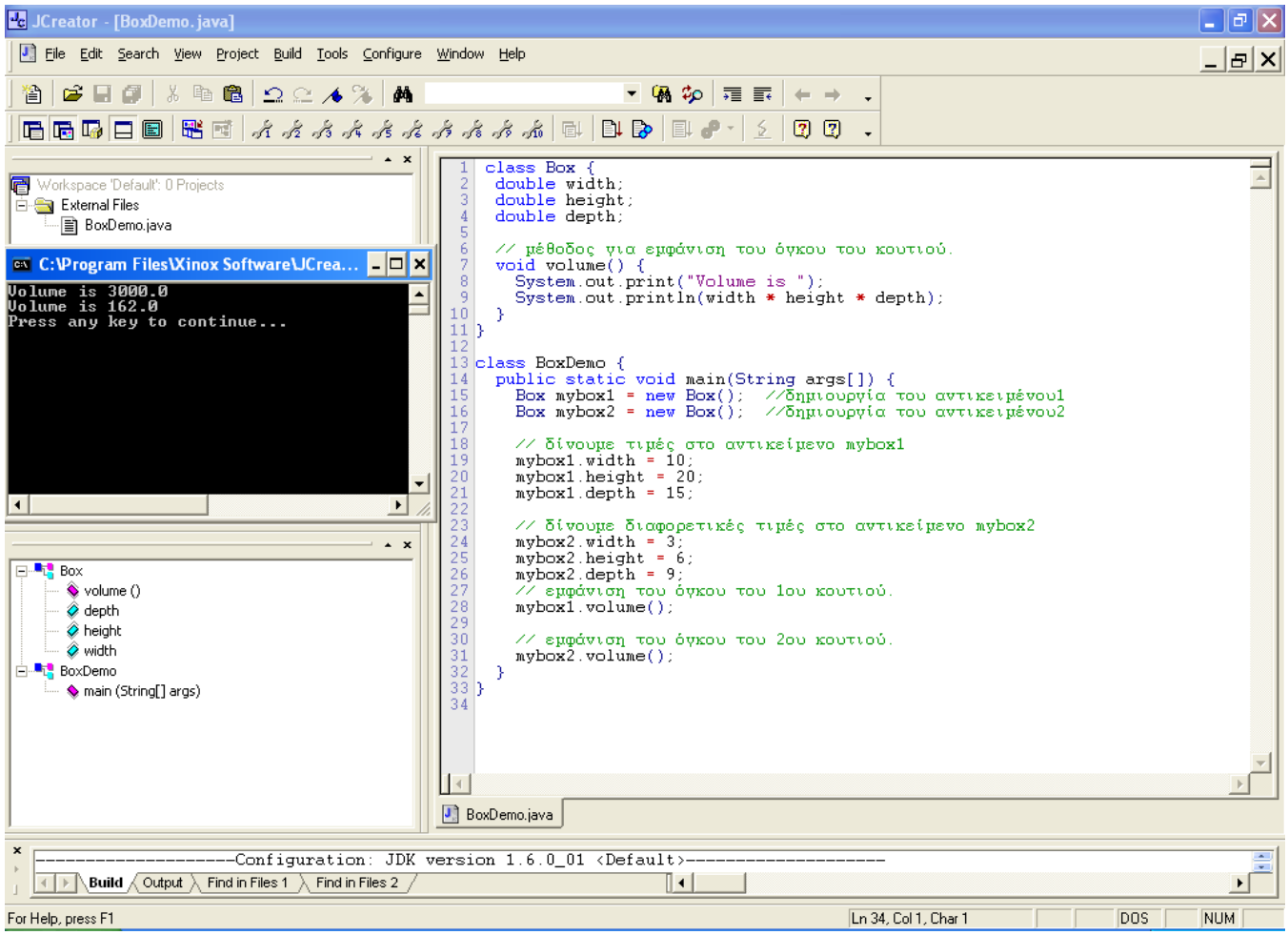
Παράδειγμα -1ο

Απλή υλοποίηση χωρίς τη χρήση δομητή – κατασκευαστή και με μέθοδο υπολογισμού χωρίς επιστρεφόμενο αποτέλεσμα.

```
class Box {

  //ορισμός των πεδίων – μεταβλητών των αντικειμένων
  double width;
  double height;
  double depth;

  // μέθοδος για εμφάνιση του όγκου του κουτιού.
  void volume() {
    System.out.print("Volume is ");
    System.out.println(width * height * depth);
  }
}

class BoxDemo {
  public static void main(String args[]) {
```

```
    Box mybox1 = new Box();  //δημιουργία του αντικειμένου1
    Box mybox2 = new Box();  //δημιουργία του αντικειμένου2

    // δίνουμε τιμές στο αντικείμενο mybox1
    mybox1.width = 10;
    mybox1.height = 20;
    mybox1.depth = 15;

    // δίνουμε διαφορετικές τιμές στο αντικείμενο mybox2
    mybox2.width = 3;
    mybox2.height = 6;
    mybox2.depth = 9;

    // εμφάνιση του όγκου του 1ου κουτιού
    mybox1.volume();

    // εμφάνιση του όγκου του 2ου κουτιού
    mybox2.volume();
  }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα *(παράθυρο Dos - μαύρη οθόνη)*:



Η κλάση Box

- Ορίζουμε τρεις μεταβλητές αντικειμένων τις width, height και depth.
- Μια μέθοδο την volume(), τύπου void γιατί δεν θα επιστρέφει τιμή αλλά απλά τη ροή του προγράμματος (δες την κλάση BoxDemo). Η μέθοδος υπολογίζει και εμφανίζει τον όγκο του αντικειμένου. Ορίσαμε τη μέθοδο volume() χωρίς λίστα παραμέτρων γιατί θέλουμε να χρησιμοποιήσει τα πεδία - μεταβλητές του αντικειμένου.

Η κλάση BoxDemo

- Μετά τον ορισμό της κλάσης μπορούμε να ορίσουμε αντικείμενα της. Εδώ ορίζουμε δύο αντικείμενα του τύπου box. Η δημιουργία των αντικειμένων γίνεται με τον τελεστή **new** ο οποίος δεσμεύει δυναμικά μνήμη για το αντικείμενο.

  Box mybox1 = new Box(); //δημιουργία του αντικειμένου1

  Box mybox2 = new Box(); //δημιουργία του αντικειμένου2

  Ο ορισμός των αντικειμένων ακολουθεί ουσιαστικά δύο βήματα: τον ορισμό της μεταβλητής που αναφέρεται (referenced) στο αντικείμενο και τη φυσική καταχώρηση του αντικειμένου στη μεταβλητή με την αντίστοιχη δέσμευση μνήμης. Στο παράδειγμά μας αναλυτικότερα η δημιουργία του αντικειμένου θα μπορούσε να γίνει με τις δύο παρακάτω εντολές:

  (1) Box mybox1;

  (2) mybox1=new Box();

  Στην (1) η mybox1 έχει τιμή μηδέν (null), ενώ στην (2) η mybox1 κρατά την διεύθυνση της μνήμης του αντίστοιχου αντικειμένου Box.

- Συντάσσουμε την μέθοδο volume() η οποία εδώ απλώς θα εμφανίσει τον όγκο των αντικειμένων mybox1 και mybox2. Η μέθοδος αυτή είναι μέλος της κλάσης Box και για τον λόγο αυτό γράφεται μέσα στο σώμα της. Συντάσσεται με τη λέξη κλειδί **void** καθότι δεν επιστρέφει κάποια τιμή σαν αποτέλεσμα, αλλά τη ροή του προγράμματος στην αμέσως επόμενη εντολή μετά την κλήση της.

```
void volume()
{
System.out.print("Volume is ");
System.out.println(width * height * depth);
}
```

Η κλήση της μεθόδου volume() γίνεται με τις εντολές :

mybox1.volume();

mybox2.volume();

Προσέξτε την χρήση της τελείας (dot) που προσδιορίζει το αντικείμενο στο οποίο αναφερόμαστε. Στο παράδειγμα η volume() εμφανίζει τον όγκο των δύο αντικειμένων (πρώτα του 1ου και μετά του 2ου).

Η ίδια μέθοδος εμφανίζει διαφορετικά αποτελέσματα για τα δύο διαφορετικά αντικείμενα (διαφορετικές δεδομένα).

Όταν μεταγλωττίσουμε το πρόγραμμα θα δημιουργηθούν δύο αρχεία με κατάληξη .class, τα **Box.class** και **BoxDemo.class**. Η java τοποθετεί την κάθε κλάση σε ξεχωριστό αρχείο.

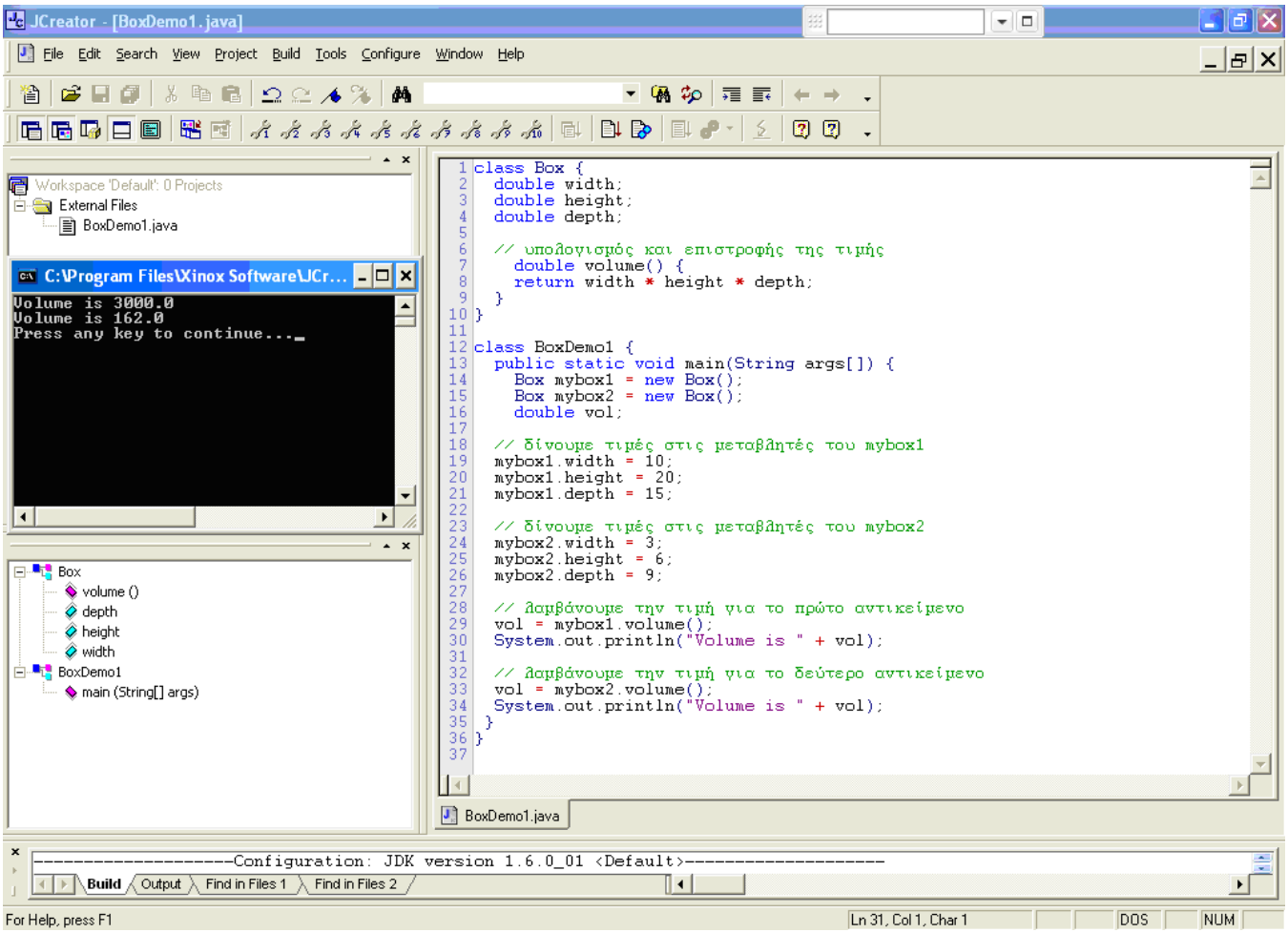
Θα μπορούσαμε να γράψουμε και εμείς τον κώδικα σε δύο ξεχωριστά πηγαία αρχεία με κατάληξη .java (και έτσι θα εργαζόμαστε συνήθως), τα **Box.java** και **BoxDemo.java**.

Παράδειγμα -2ο

Απλή υλοποίηση χωρίς τη χρήση δομητή – κατασκευαστή αλλά με μέθοδο υπολογισμού με επιστρεφόμενο αποτέλεσμα.

```
class Box {

  //ορισμός των πεδίων - μεταβλητών
  double width;
  double height;
  double depth;

   // υπολογισμός και επιστροφής της τιμής
   double volume() {
   return width * height * depth;
  }
}

class BoxDemo1 {
  public static void main(String args[]) {
    Box mybox1 = new Box();
    Box mybox2 = new Box();
    double vol;

  // δίνουμε τιμές στις μεταβλητές του mybox1
  mybox1.width = 10;
  mybox1.height = 20;
  mybox1.depth = 15;

  // δίνουμε τιμές στις μεταβλητές του mybox2
  mybox2.width = 3;
  mybox2.height = 6;
  mybox2.depth = 9;

  // λαμβάνουμε την τιμή για το πρώτο αντικείμενο
  vol = mybox1.volume();
  System.out.println("Volume is " + vol);

  // λαμβάνουμε την τιμή για το δεύτερο αντικείμενο
  vol = mybox2.volume();
  System.out.println("Volume is " + vol);
 }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα *(παράθυρο Dos - μαύρη οθόνη)*:



Η κλάση Box

- Ορίζει τρεις μεταβλητές-πεδία αντικειμένων (που θα λάβουν διαφορετικές τιμές για κάθε αντικείμενο)
- Μία μέθοδο υπολογισμού του όγκου του κάθε αντικειμένου – box. Η μέθοδος αυτή θα υπολογίζει τον όγκο του κάθε αντικειμένου box (width * height * depth) και θα τον επιστρέφει (return) στη μεταβλητή vol της κλάσης BoxDemo1. Ορίσαμε τη μέθοδο volume() χωρίς λίστα παραμέτρων (θα χρησιμοποιήσει τις μεταβλητές αντικειμένου) και με τύπο double γιατί το επιστρεφόμενο αποτέλεσμα θα είναι τύπου double.

Η κλάση BoxDemo1

- Δημιουργεί δύο αντικείμενα τύπου Box – το mybox1 και το mybox2. Τα αντικείμενα mybox1 και mybox2 λαμβάνουν τιμές (10, 20 και 15 και 3, 6, 9 αντίστοιχα) με ξεχωριστές εντολές για τις μεταβλητές width, height και depth.
- Στο παράδειγμα τροποποιήσαμε την μέθοδο volume(), καθώς και τις εντολές κλήσης της, έτσι ώστε να πάρουμε σαν επιστρεφόμενη τιμή τον όγκο του κάθε αντικειμένου box:

```
double volume()
```

```
    {
      return width * height * depth;
    }
```

Προσέξτε την σύνταξη των εντολών κλήσης της μεθόδου:

vol = mybox1.volume();

vol = mybox2.volume();

Στο παράδειγμα η μεταβλητή vol είναι αυτή που θα λάβει το αποτέλεσμα της μεθόδου volume(). Η μεταβλητή αυτή πρέπει να είναι του ίδιου τύπου με το επιστρεφόμενο από την μέθοδο αποτέλεσμα. Η λήψη της επιστρεφόμενης τιμής μπορεί να εμφανιστεί και απ' ευθείας με την εντολή εμφάνισης του αποτελέσματος :

System.out.println("Volume is " + mybox1.volume());

Παράδειγμα -3ο

Υλοποίηση με τη χρήση παραμετρικό δομητή – κατασκευαστή και μέθοδο υπολογισμού με επιστρεφόμενη τιμή.

```
class Box {
  double width;
  double height;
  double depth;

  // ο δομητής του Box.
  Box(double x, double y, double z) {
    System.out.println("Constructing Box");
    width = x;
    height = y;
    depth = z;
  }

  // υπολογισμός και επιστροφή του όγκου
   double volume() {
    return width * height * depth;
  }
}

class BoxDemo {

  public static void main(String args[]) {

    // δημιουργία και αρχικοποίηση αντικειμένων
   Box mybox1 = new Box(10, 20, 15);
   Box mybox2 = new Box(3, 6, 9);

    double vol;

    // λαμβάνεται ο όγκος του πρώτου box
    vol = mybox1.volume();
    System.out.println("Volume is " + vol);
```

Τα **πεδία – μεταβλητές** των αντικειμένων: μήκος, ύψος και πλάτος.

Ο **δομητής ή κατασκευαστής** των αντικειμένων, αρχικοποιεί τα αντικείμενα. Δέχεται τρεις τιμές με τις οποίες αρχικοποιεί κάθε αντικείμενο τύπου Box.

Η **μέθοδος** υπολογισμού του όγκου του αντικειμένου – box. Ο όγκος του αντικειμένου αφού υπολογιστεί, επιστρέφεται στο κυρίως πρόγραμμα (μεταβλητή vol).

Η κλάση Δοκιμής BoxDemo

Δημιουργία και **αρχικοποίηση** των δύο αντικειμένων

Ορισμός μεταβλητής για τη λήψη της τιμής του όγκου

Λήψη και **εμφάνιση** του όγκου του 1ου – box (3000.0)

```
    // λαμβάνεται ο όγκος του δευτέρου box
    vol = mybox2.volume();
    System.out.println("Volume is " + vol);
  }
}
```

Λήψη και **εμφάνιση** του όγκου του 2ου – box (162.0)

Η κλάση Box

- Ορίζει τρεις μεταβλητές-πεδία αντικειμένων (που θα λάβουν διαφορετικές τιμές για κάθε αντικείμενο)
- Ένα δομητή-κατασκευαστή αντικειμένων που έχει το ίδιο όνομα με αυτό της κλάσης. Για να έχουμε διαφορετικές τιμές για τα αντικείμενα όπως είναι το σωστό, παραμετροποιούμε τον δομητή. Οι μεταβλητές x, y, και z είναι τοπικές (ισχύουν αποκλειστικά μέσα στο σώμα του δομητή - θα εξηγήσουμε τις τοπικές μεταβλητές και τη διάρκεια ζωής των αναλυτικότερα παρακάτω).
- Μία μέθοδο υπολογισμού του όγκου του κάθε αντικειμένου – box. Η μέθοδος αυτή θα υπολογίζει τον όγκο του κάθε box (τύπου double - width * height * depth) και θα τον επιστρέφει (return) στη μεταβλητή vol.

Η κλάση BoxDemo

- Δημιουργεί και αρχικοποιεί δύο αντικείμενα Box – το mybox1 και το mybox2. Το mybox1 έχει τιμές 10, 20 και 15 για τις μεταβλητές width, height και depth, ενώ το mybox2 τις τιμές 3, 6, 9 αντίστοιχα. Οι εντολές κλήσης του δομητή – κατασκευαστή για την δημιουργία και αρχικοποίηση των δύο αντικειμένων γίνεται με τις εντολές:

```
Box mybox1 = new Box(10, 20, 15);
Box mybox2 = new Box(3, 6, 9);
```

Προσοχή στη σύνταξη:

Πρώτα με τη λέξη Box καθορίζεται ο τύπος του αντικειμένου mybox1. Μετά με την κλήση του τελεστή **new** καλείται ο δομητής - κατασκευαστής για να δημιουργήσει και να αρχικοποιήσει το αντικείμενο σύμφωνα με τις τιμές που του στέλνουμε σαν παραμέτρους. Κάθε μία από αυτές τις δύο εντολές είναι ουσιαστικά μια διαδικασία δύο βημάτων (δύο ξεχωριστών εντολών):

```
Box mybox1;          //με το όνομα mybox1 θα εννοείται ένα αντικείμενο του τύπου Box
mybox1 = new Box();  //δημιουργείται το αντικείμενο mybox1 όπως ορίζεται στην Box
```

Αν δεν είχαμε τον δομητή-κατασκευαστή οι τιμές θα έπρεπε να δηλωθούν αναλυτικά με τη χρήση του ονόματος του αντικειμένου, της τελείας (.) και της τιμής που θα πάρει κάθε μεταβλητή, δηλαδή:

```
mybox1. width = 10;
mybox1. height = 20;
mybox1.depth = 15;
```

Ας δούμε άλλα δύο παραδείγματα το ένα χωρίς – και το άλλο με τη χρήση δομητή - κατασκευαστή.

Στο παρακάτω παράδειγμα υποθέτουμε ότι ένα κατάστημα εμπορεύεται τρία είδη προϊόντων (product1, product2, product3) με τιμές αντίστοιχα: 10.0, 7.5 και 5.5 Ευρώ, ανά τεμάχιο. Αν **N** πελάτες αγοράσουν quantity1, quantity2, quantity3 τεμάχια από τα τρία είδη προϊόντων, ζητείται να υπολογισθεί και να εμφανισθεί η τελική που θα πληρώσει ο κάθε πελάτης. Επειδή το κατάστημα έχει πολλούς πελάτες το πρόγραμμα θα ορίζει ένα γενικό τρόπο υπολογισμού της τελικής τιμής και θα εξειδικεύει τον υπολογισμό για τους διάφορους πελάτες.

Παράδειγμα – 4ο: Χωρίς τη χρήση δομητή

```
class Products {
        // Definition of class variables
        static double product1 = 10.0;
        static double product2 =  7.5;
        static double product3 =  5.5;

        int quantity1, quantity2, quantity3;
        double total_price;

        // Definition of method Calculate Price
        void calculate_price() {
          total_price = product1*quantity1 + product2*quantity2 + product3*quantity3;
          System.out.println("Total price " + total_price);
        }
}
class DemoProducts {
        public static void main(String args[]) {

        // We create two new objects of type Products
        Products pro1 = new Products();
        Products pro2 = new Products();

        // We give values to instance variables
        pro1.quantity1 = 3;
        pro1.quantity2 = 1;
        pro1.quantity3 = 4;
        pro2.quantity1 = 5;
        pro2.quantity2 = 2;
        pro2.quantity3 = 3;

        // Using calculate_price() we obtain the two total prices
        pro1.calculate_price();
        pro2.calculate_price();
        }
}
```

Τα αποτελέσματα του προγράμματος:

```
Total price 59.5
```

```
Total price 81.5
```

Στην κλάση Products ορίζονται οι μεταβλητές και η μέθοδος υπολογισμού της τελικής τιμής που θα πληρώνει ο κάθε πελάτης. Δίνονται οι μεταβλητές κλάσης product1, product2, product3 (ίδιες για όλα τα αντικείμενα), ενώ δεν δίνονται οι τιμές των μεταβλητών των αντικειμένων (που διαφέρουν από αντικείμενο σε αντικείμενο). Στην κλάση DemoProducts μετά τη δήλωση της main(), θα δημιουργήσουμε δύο αντικείμενα της κλάσης Products, θα δώσουμε τιμές στις *μεταβλητές αντικειμένων* (ποσότητες) και τέλος θα καλέσουμε την μέθοδο calculate_price για να υπολογίσουμε και εμφανίσουμε την τελική τιμή για τα δύο αντικείμενα.

Παράδειγμα – 5ο: Με τη χρήση δομητή και μέθοδο υπολογισμού της τελικής τιμής με επιστρεφόμενη τιμή

```
class Products1 {

        static double product1 = 10.0;
        static double product2 =  7.5;

        static double product3 =  5.5;

        int quantity1, quantity2, quantity3;
        double total_price;

        // Definition of constructor.
        Products1 (int a, int b, int c)
        {
         quantity1 = a;
         quantity2 = b;
         quantity3 = c;
        }
        // Definition of method Calculate Price.
        double calculate_price()
        {
          total_price = product1*quantity1 + product2*quantity2 + product3*quantity3;
         return total_price;
          }
    }

class DemoProducts1 {
        public static void main(String args[]) {

        // We create two new objects of type Products and give values.
        Products1 pro1 = new Products1(3,1,4);
        Products1 pro2 = new Products1(5,2,3);

        // Using calculate_price() we obtain the two total prices.
        double tot_amount;

        tot_amount=pro1.calculate_price();
        System.out.println("Total price-1 = "+tot_amount);
```

```
        tot_amount=pro2.calculate_price();
        System.out.println("Total price-2 = "+tot_amount);
        }
}
```

Τα αποτελέσματα του προγράμματος:

```
Total price-1 = 59.5
Total price-2 = 81.5
```

Με τα παραδείγματα αυτά είδαμε τη χρήση του δομητή-κατασκευαστή και των μεθόδων. Επίσης είδαμε πως καλούμε τον δομητή-κατασκευαστή αλλά και οποιαδήποτε άλλη μέθοδο με την χρήση παραμέτρων. Για τους τύπους μεθόδων, την κλήση τους, τις παραμέτρους και την επιστροφή τιμής θα αναφερθούμε αναλυτικότερα παρακάτω.

Μια δεύτερη ματιά στους Δομητές – Κατασκευαστές *(Constructors)*

Ο συντομότερος και αποτελεσματικότερος τρόπος αρχικοποίησης των μεταβλητών είναι να γίνεται κατά την στιγμή της δημιουργίας του αντικειμένου και όχι με τον τρόπο που είδαμε στα δύο πρώτα παραδείγματα. Ο δομητής:

- αρχικοποιεί αμέσως το αντικείμενο κατά την στιγμή της δημιουργίας του
- έχει το ίδιο όνομα με το όνομα της κλάσης στην οποία ανήκει το αντικείμενο
- συντάσσεται με τον ίδιο τρόπο όπως μία μέθοδος
- δεν επιστρέφει αποτέλεσμα, όμως δεν περιλαμβάνει στην σύνταξή του τη λέξη κλειδί **void**

Κάθε κλάση μπορεί να περιέχει πολλούς διαφορετικούς δομητές - με διαφορετικές παραμέτρους για αρχικοποίηση αντικειμένων με διαφορετικές τιμές. Επίσης μπορεί να μην περιέχει κανένα δομητή, όμως σε αυτή την περίπτωση ισχύει ο αρχικός (default) - νοητός δομητής που αρχικοποιεί με μηδενικές τιμές τις μεταβλητές.

Μια δεύτερη ματιά στις μεθόδους

Όπως είδαμε οι μέθοδοι είναι εκείνα τα τμήματα του κώδικα όπου εκτελούνται οι ουσιαστικές εργασίες του προγράμματος. Η γενική μορφή μιας μεθόδου είναι:

```
<τύπος>  <όνομα μεθόδου> (λίστα παραμέτρων)
{
  //κώδικας - σώμα της μεθόδου
}
```

Ο <τύπος> της μεθόδου μπορεί να είναι ένας από τους οκτώ βασικούς τύπους της Java ή κάποιος δικός μας και καθορίζει τον τύπο της επιστρεφόμενης τιμής (του αποτελέσματος). Όταν η μέθοδος

επιστρέφει τιμή συντάσσεται με την **return <τιμή/μεταβλητή>;** ενώ όταν δεν επιστρέφει κάποια τιμή, τότε δεν έχει τύπο και συντάσσεται με την λέξη κλειδί **void.**

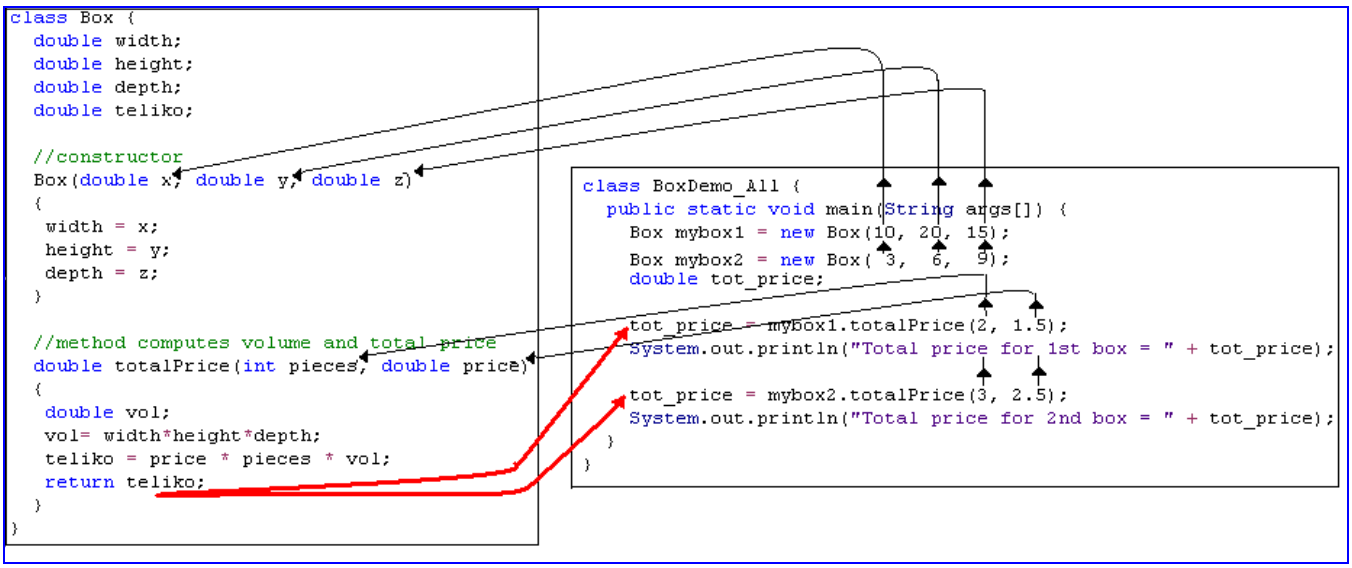
Η λίστα παραμέτρων είναι μια ακολουθία από μεταβλητές με τους τύπους της, χωρισμένες με κόμμα. Είναι **τοπικές μεταβλητές** (local variables), δηλαδή ισχύουν μόνο όσο εκτελείται η συγκεκριμένη μέθοδος. Αυτό σημαίνει ότι τα ίδια ονόματα παραμέτρων μπορούν να χρησιμοποιηθούν και σε άλλες μεθόδους. Αν η μέθοδος δεν έχει παραμέτρους, αφήνουμε κενές τις παρενθέσεις. Μπροστά από κάθε μεταβλητή συντάσσεται ο τύπος της που προκαθορίζει τον τύπο της τιμής που θα λάβει όταν κληθεί από την αντίστοιχη εντολή. Η εντολή κλήσης περιέχει τις τιμές που θα "περάσουν" αντίστοιχα στις παραμέτρους της μεθόδου. Οι κλήσεις των μεθόδων είναι γνωστές και ως **μηνύματα** (messages).

Μια δεύτερη ματιά στο πέρασμα των παραμέτρων και στις επιστρεφόμενες τιμές

Οι παράμετροι που χρησιμοποιούνται κατά την κλήση μιας μεθόδου, ανάλογα με την χρήση των, ονομάζονται είτε **παράμετροι τιμής** *(call – by - value),* ή **παράμετροι αναφοράς** *(call – by – reference)*. Οι παράμετροι τιμής, που χρησιμοποιούνται μόνο για είσοδο τιμών, περνούν απλώς την τιμή τους στις αντίστοιχες παραμέτρους της μεθόδου. Οποιαδήποτε μεταβολή στην τιμή τέτοιων παραμέτρων μέσα στη μέθοδο δεν αντανακλάται στις αντίστοιχες παραμέτρους κλήσης.

Οι παράμετροι αναφοράς, είναι οι παράμετροι που περνούν στη μέθοδο μία σχέση απλής αναφοράς και όχι την ίδια την τιμή τους. Με τον τρόπο αυτό μια αλλαγή στην τιμή των, μέσα στη μέθοδο, αντανακλάται άμεσα και στις αναφερόμενες παραμέτρους της κλήσης.

Οι βασικοί τύποι της Java περνούν σαν παράμετροι τιμής, ενώ τα αντικείμενα σαν παράμετροι αναφοράς. Σε αντίθεση με άλλες γλώσσες προγραμματισμού, η Java δεν επιτρέπει να περαστεί κάποια μέθοδος σαν παράμετρος σε άλλη μέθοδο. Αυτό που επιτρέπεται είναι να περαστεί σαν παράμετρος το αντικείμενο και στη συνέχεια να κληθεί η επιθυμητή μέθοδος του αντικειμένου.



Το πρόγραμμα *(πλήρης δομητής – μέθοδος με παραμέτρους):*

```
class Box {
```

```
  double width;
  double height;
  double depth;
  double teliko;

  //constructor
  Box(double x, double y, double z)  {
   width = x;
   height = y;
   depth = z;
  }
  //method computes volume and total price
  double totalPrice(int pieces, double price)   {
   double vol;
   vol= width*height*depth;
   teliko = price * pieces * vol;
   return teliko;
  }
}
class BoxDemo_All {
  public static void main(String args[]) {
    Box mybox1 = new Box(10, 20, 15);
    Box mybox2 = new Box(3, 6, 9);
    double tot_price;

    tot_price = mybox1.totalPrice(2, 1.5);
    System.out.println("Total price for 1st box = " + tot_price);

    tot_price = mybox2.totalPrice(3, 2.5);
    System.out.println("Total price for 2nd box = " + tot_price);
  }
}
```

Στο παρακάτω παράδειγμα θα δούμε την χρήση των παραμέτρων τιμών. Μετά την κλήση της μεθόδου, αν και εσωτερικά αλλάζουν οι τιμές, οι αντίστοιχες παράμετροι τιμής δεν επηρεάζονται.

```
class Test {
   void meth(int i, int j) {
     i *= 2;
     j /= 2;
   }
}

class CallByValue {
  public static void main(String args[]) {
    Test ob = new Test();
    int a = 15, b = 20;
    System.out.println("a and b before call: " + a + " " + b);
    ob.meth(a, b);
    System.out.println("a and b after call: " + a + " " + b);
  }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα :

```
a and b before call : 15  20
a and b after call : 15  20
```

Προσοχή στις Static μεταβλητές – και μεθόδους:

Αν θέλουμε μία μεταβλητή ή μέθοδος να είναι κοινή για όλα τα αντικείμενα της κλάσης, τότε την δηλώνουμε με την λέξη κλειδί static. Σε μια τέτοια δήλωση μεταβλητής ή μεθόδου έχουμε πρόσβαση με τη χρήση του ονόματος της κλάσης (Δες παραδείγματα 4ο και 5ο). Αν δηλώσουμε μία μεταβλητή static, τότε αυτή δημιουργείται μόλις δηλωθεί το πρώτο αντικείμενο της κλάσης, ο δε χώρος δεσμεύεται μία φορά και χρησιμοποιείται για όλα τα αντικείμενα της κλάσης. Π.χ.
static float syntelestis_FPA; Αν θέλουμε να αναφερθούμε σε αυτήν την μεταβλητή μπορούμε απ' ευθείας μέσω του ονόματος της κλάσης, π.χ. Ypologismos. syntelestis_FPA = 0.19;
Τα ίδια ισχύουν και σε μια τέτοια δήλωση μεθόδου, π.χ. static setFPA(float s_FPA)
Ypologismos.setFPA(0.19);
Μέθοδοι που δηλώνονται static έχουν τους παρακάτω περιορισμούς:

- Μπορούν να καλούν μόνο στατικές μεθόδους
- Μπορούν να έχουν πρόσβαση μόνο σε στατικά δεδομένα
- Δεν χρησιμοποιούν τους τελεστές **this** και **super** (θα τους δούμε αναλυτικότερα παρακάτω)

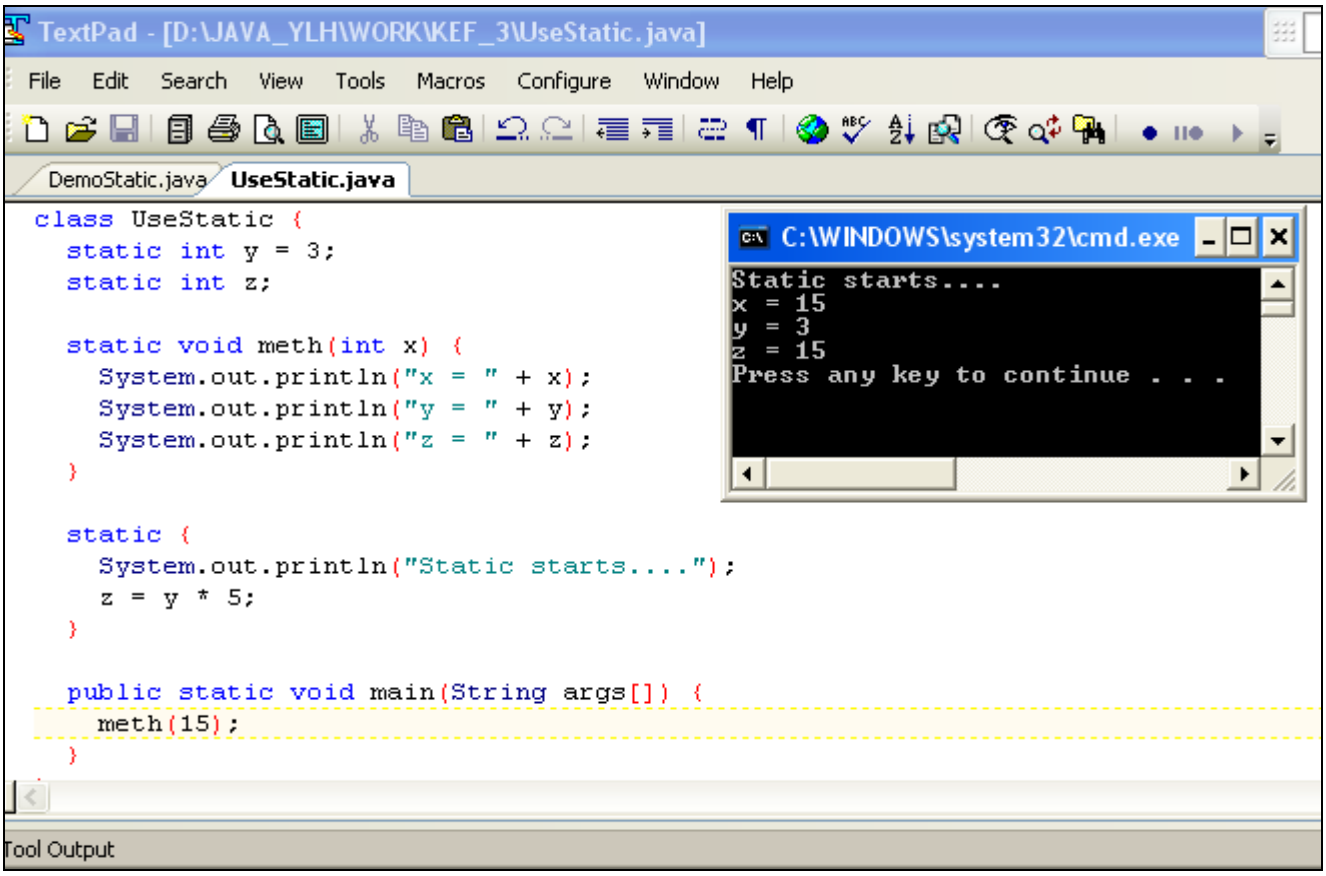
Παράδειγμα – 1ο:

```
class StaticUse  {
 static int a = 15;
 static int b = 20;

static void callStatic() {
 System.out.println("a = "+a);
}
}

class DemoStatic  {
 public static void main (String[] args)  {
  StaticUse.callStatic();
  System.out.println("b = "+StaticUse.b);
}
}
```

Το αποτέλεσμα του προγράμματος:

```
a = 15
b = 20
```

Παράδειγμα – 2ο:



Έλεγχος Προσπέλασης (access control)

Τρεις δεσμευμένες λέξεις (κλειδιά) που χρησιμοποιούμε για τους ορισμούς των μελών μιας κλάσης επιτρέπουν ή απαγορεύουν την προσπέλαση οι: **public, private** και **protected**. Μία κλάση μπορεί να συνταχθεί χωρίς χαρακτηριστικό προσβασιμότητας, οπότε είναι προσπελάσιμη από τις κλάσεις του **πακέτου** (package - πλήθος κλάσεων μιας εφαρμογής) στο οποίο ανήκει. Μπορεί όμως να χαρακτηριστεί ως public, οπότε μπορεί να προσπελαστεί από οπουδήποτε (είτε μέσα από το πακέτο στο οποίο ανήκει, είτε έξω από αυτό). Η main() ορίζεται πάντα public για να καλείται έξω από το πρόγραμμά δηλαδή, από το σύστημα run-time της java. Κάθε ορισμός που δεν περιλαμβάνει ένα από τους ανωτέρω ελέγχους θεωρείται public. Για τα μέλη της κλάσης (μεταβλητές, μεθόδους) ισχύουν οι εξής επιλογές προσπέλασης:

Χαρακτηρισμός	Βαθμός προσπέλασης
public	Από οπουδήποτε
κανένας	Μόνο από την ίδια κλάση ή το ίδιο πακέτο
private	Μόνο από την ίδια κλάση
protected	Από την ίδια κλάση, ίδιο πακέτο, ή υποκλάση της δεδομένης κλάσης σε άλλο πακέτο

Γνωρίζοντας αυτές τις ιδιότητες μπορούμε να επιλέγουμε κάθε φορά το χαρακτηριστικό πρόσβασης – προσπέλασης που επιθυμούμε ανάλογα με τις ανάγκες του προγράμματος.

Private

Ο πιο απαγορευτικός χαρακτηρισμός πρόσβασης είναι ο private. Ένα μέλος που χαρακτηρίζεται private είναι προσβάσιμο μόνο από τα άλλα μέλη της κλάσης στην οποία ορίζεται και όχι από τα μέλη άλλων κλάσεων ή υποκλάσεων (θα τις δούμε σε άλλο μάθημα). Τα private μέλη είναι σαν μυστικά που δε θέλουμε να τα μάθει κανείς. Για να δηλώσουμε ένα private μέλος, χρησιμοποιούμε τη δεσμευμένη λέξη private στη δήλωσή του. Συνήθως οι μεταβλητές των αντικειμένων δηλώνονται ως private, ενώ οι μέθοδοι πρόσβασης ως public. Στο παρακάτω παράδειγμα δηλώνουμε μία private μεταβλητή και μία private μέθοδο:

```
class Test1 {
   private int var1private;
   private void privateMethod1() {
      System.out.println("privateMethod");
   }
}
```

Αντικείμενα του τύπου Test1 μπορούν να τροποποιήσουν την μεταβλητή var1private και μπορούν να καλέσουν την μέθοδο privateMethod(), αλλά αντικείμενα διαφορετικού τύπου δεν μπορούν. Για παράδειγμα, η κλάση Test2:

```
class Test2 {
   void accessMethod() {
      Test1 a = new Test1();
      a.var1private = 10;
      a.privateMethod();
   }
}
```

δεν έχει πρόσβαση στην μεταβλητή var1private ή στην μέθοδο privateMethod() των αντικειμένων τύπου Test1.

Ο μεταγλωττιστής θα τυπώσει το παρακάτω μήνυμα και θα αρνηθεί να συνεχίσει:

```
Beta.java:9: Variable var1private in class Test1 not accessible
from class Test2.
        a.var1private = 10;
         ^
1 error
```

Protected

Ένα μέλος που χαρακτηρίζεται protected είναι προσβάσιμο από τα μέλη της αντίστοιχης κλάσης, τις υποκλάσεις της και σε όλες τις κλάσεις του ίδιου πακέτου (πλήθος κλάσεων μιας εφαρμογής). Χρησιμοποιείται ιδιαίτερα όταν τα μέλη των υποκλάσεων χρειάζεται να έχουν πρόσβαση σε αυτά τα μέλη. Τα protected μέλη είναι σαν τα οικογενειακά μυστικά-δεν μας ενδιαφέρει αν το ξέρει η οικογένεια και μερικοί φίλοι, αλλά όχι και οι ξένοι. Για να δηλώσουμε ένα protected μέλος, χρησιμοποιούμε τη δεσμευμένη λέξη protected. Αν η κλάση είναι υποκλάση μίας κλάσης και βρίσκεται στο ίδιο πακέτο με την κλάση, τότε έχει πρόσβαση στα protected μέλη της υπερκλάσης της. Η protected αφορά την κληρονομικότητα.

Στο παρακάτω παράδειγμα θα δούμε την χρήση του ελέγχου προσπέλασης και τις διαφορές μεταξύ public και private.

```
class Test {
  int a;                  // χωρίς δήλωση, άρα public
  public int b;           // public προσπέλαση
  private int c;          // private προσπέλαση

  // μέθοδοι για προσπέλαση της c
  void setc(int i) {  // ορισμός της τιμής της c
    c = i;
  }
  int getc() {          // λήψη της τιμής της c
    return c;
  }
}
class AccessTest {
  public static void main(String args[]) {
    Test ob = new Test();

    // Οι a και b μπορούν να προσπελαθούν απ' ευθείας
    ob.a = 10;
    ob.b = 20;

// ο παρακάτω ορισμός είναι λάθος
   ob.c = 100;                  //λάθος στην προσπέλαση

    // προσπέλαση στη c μόνο μέσω της μεθόδου της
    ob.setc(100);               // OK

    System.out.println("a, b, and c: " + ob.a + " " + ob.b + " " + ob.getc());
  }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα :

a, b, and c : 10 20 100

Η c δεν μπορεί να προσπελαθεί από κώδικα έξω από την κλάση της, αλλά μόνο μέσα από τις public – μεθόδους **setc()** και **getc()**.

Οι μέθοδοι αλλαγής τιμών ή τοποθέτησης - (set) και πρόσβασης ή απόκτησης τιμής - (get)

Για να έχουμε πρόσβαση (τοποθετήσουμε ή αλλάξουμε τιμή, ή να αποκτήσουμε την υπάρχουσα τιμή) στις private μεταβλητές θα πρέπει να χρησιμοποιήσουμε τις κατάλληλες μεθόδους τύπου public, **set *(mutator)*** και **get *(accessor)***. Οι μέθοδοι τοποθέτησης ή αλλαγής τιμής – set είναι μέθοδοι τύπου public που ορίζουν ή αλλάζουν τις τιμές των private μεταβλητών, ενώ οι μέθοδοι απόκτησης τιμής – get είναι μέθοδοι τύπου public που εμφανίζουν τις τιμές των private μεταβλητών. Συνήθως, τα ονόματα αυτών των μεθόδων προέρχονται από το όνομα της μεταβλητής με την προσθήκη της set και get. Για παράδειγμα, αν το όνομα της μεταβλητής είναι syntelestis, τότε τα ονόματα των μεθόδων μπορούν να γραφούν ως: setSyntelestis() και getSyntelestis() ή καλύτερα set_ Syntelestis() και get_ Syntelestis().

Το παράδειγμα με της κλάσης Box θα μπορούσε να είναι της παρακάτω μορφής:

```
public class Box
{
  // fields
  private int length;
  private int width;
  private int height;

  // methods
  public void setLength(int p)
  {length = p;}


public void setWidth(int p)
  {width = p;}

public void setHeight(int p)
  {height = p;}

  public int displayVolume()
  {System.out.println(length*width*height);}
}
```

Υπερφόρτωση Μεθόδων (Overloading Methods)

Στη java είναι δυνατόν να ορίσουμε μέσα σε μία κλάση δύο ή περισσότερες μεθόδους με το ίδιο όνομα αλλά με διαφορετικές παραμέτρους. Αυτό ονομάζεται υπερφόρτωση μεθόδων και στην περίπτωση αυτή ο τύπος των δεδομένων και ο αριθμός των παραμέτρων λαμβάνονται υπόψη στον προσδιορισμό της μεθόδου που θα εκτελεστεί. Στο παρακάτω παράδειγμα θα δούμε πως υλοποιείται η υπερφόρτωση μεθόδων.

```
class OverloadDemo {
  void test() {
    System.out.println("No parameters");
  }

  // υπερφόρτωση με μία integer παράμετρο
  void test(int a) {
    System.out.println("a: " + a);
  }

  // υπερφόρτωση με δύο integer παραμέτρους
  void test(int a, int b) {
    System.out.println("a and b: " + a + " " + b);
  }

  // υπερφόρτωση με μία double παράμετρο
  double test(double a) {
    System.out.println("double a: " + a);
    return a*a;
  }
}

class Overload {
  public static void main(String args[]) {
    OverloadDemo ob = new OverloadDemo();
    double result;

    // κλήση όλων των παραλαγών της test()
    ob.test();
    ob.test(10);
    ob.test(10, 20);
    result = ob.test(123.2);
    System.out.println("Result of ob.test(123.2): " + result);
  }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα :

```
No parameters
a: 10
a and b: 10 20
double a: 123.2
Result of ob.test(123.2): 15178.240000000002
```

Για την επιλογή της κατάλληλης μεθόδου η java μπορεί να κάνει και αυτόματη μετατροπή τύπου δεδομένων. Στο παρακάτω παράδειγμα θα κάνει μετατροπή του τύπου integer σε double έτσι ώστε να γίνει η κλήση της κατάλληλης μεθόδου.

```
class OverloadDemo {
  void test() {
    System.out.println("No parameters");
  }


  // υπερφόρτωση με δύο integer παραμέτρους
```

```
  void test(int a, int b) {
    System.out.println("a and b: " + a + " " + b);
  }

  // υπερφόρτωση με double παράμετρο
  void test(double a) {
    System.out.println("Inside test(double) a: " + a);
  }
}

class Overload {
  public static void main(String args[]) {
    OverloadDemo ob = new OverloadDemo();
    int i = 88;
    ob.test();
    ob.test(10, 20);

    ob.test(i);          // θα καλέσει την test(double). Παράδοξο;
    ob.test(123.2);      // θα καλέσει την test(double)
  }
}
```

Στο ανωτέρω παράδειγμα επειδή δεν υπάρχει μέθοδος test(int) η java μετατρέπει τον τύπο integer σε double και καλεί με υπερφόρτωση την κατάλληλη μέθοδο που είναι η test(double). Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα :

```
No parameters
a and b: 10 20
Inside test(double) a: 88.0
Inside test(double) a: 123.2
```

Υπερφόρτωση (overloading) Δομητών

Υπερφόρτωση μπορεί να εφαρμοσθεί και στους δομητές όπως και στις απλές μεθόδους. Για παράδειγμα, κατά τον ορισμό του αντικειμένου box θα συμβεί υπερφόρτωση δομητών αν χρησιμοποιήσουμε διαφορετικές παραμέτρους. Στο παρακάτω παράδειγμα θα δούμε πως μπορεί συμβεί υπερφόρτωση όταν ο ορισμός του αντικειμένου box μπορεί να γίνει με την χρήση διαφορετικών δομητών.

```
class Box {
  double width;
  double height;
  double depth;

  // δομητής που χρησιμοποιείται για όλες τις διαστάσεις
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }
  //δομητής όταν δεν ορίζουμε κάποια διάσταση
```

```
  //χρησιμοποιούμε τις τιμές –1
  Box() {
    width = -1;
    height = -1;
    depth = -1;
  }
  // δομητής που χρησιμοποιείται όταν ορίζεται το box
  Box(double len) {
    width = height = depth = len;
  }

  // υπολογισμός και επιστροφή του όγκου
  double volume() {
    return width * height * depth;
  }
}

class OverloadCons {
  public static void main(String args[]) {


    // δημιουργία του box με διαφορετικούς δομητές
    Box mybox1 = new Box(10, 20, 15);
    Box mybox2 = new Box();
    Box mycube = new Box(7);
    double vol;

    // λήψη του όγκου του 1ου box
    vol = mybox1.volume();
    System.out.println("Volume of mybox1 is " + vol);

    // λήψη του όγκου του 2ου box.
    vol = mybox2.volume();
    System.out.println("Volume of mybox2 is " + vol);

    // λήψη του όγκου του κύβου.
    vol = mycube.volume();
    System.out.println("Volume of mycube is " + vol);
  }
}
```

Αν εκτελέσουμε το πρόγραμμα θα πάρουμε τα παρακάτω αποτελέσματα :

```
Volume of mybox1 is 3000.0
Volume of mybox2 is –1.0
Volume of mycube is 343.0
```