ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Εισαγωγή στον Αντικειμενοστραφή προγραμματισμό

Αθανάσιος Ανδρούτσος

Σύνθετες Δομές Δεδομένων

- Εκτός από τους Πίνακες και τα Strings, η Java παρέχει ένα ακόμα **σύνθετο τύπο δεδομένων**, τον τύπο **Class**. Πρόκειται για σύνθετο τύπο δεδομένων που μπορεί να περιλαμβάνει:

  1. *Δεδομένα διαφόρων τύπων* (όχι μόνο ενός τύπου όπως οι πίνακες),
  2. *Μεθόδους*

- Τα **Δεδομένα** και οι **Μέθοδοι** μιας κλάσης ονομάζονται ***μέλη της κλάσης***

Κλάσεις (1)

- Ο λόγος που εισήχθησαν οι **Κλάσεις** στον Προγραμματισμό είναι για να μπορούμε να απεικονίσουμε σύνθετα αντικείμενα του **πραγματικού κόσμου**
- Στον πραγματικό κόσμο, κάθε αντικείμενο αποτελείται από **δεδομένα** και **λειτουργίες.**
- Για παράδειγμα, ένα μεταφορικό όχημα αποτελείται από ***δεδομένα*** (όπως τύπος, έτος κατασκευής, κυβισμός, κλπ.) καθώς και ***λειτουργίες*** (ξεκίνα, στρίψε δεξιά, σταμάτα, άναψε φώτα, κλπ.)

Κλάσεις (2)

- Τα δεδομένα των κλάσεων ονομάζονται **πεδία (fields)** και οι λειτουργίες ονομάζονται **μέθοδοι (methods)**
- Έτσι για παράδειγμα, ένα **μεταφορικό όχημα** μπορεί να αναπαρασταθεί από μία κλάση, για παράδειγμα **Vehicle** η οποία αποτελείται από ***πεδία*** όπως **String type**, **int releaseYear**, **int cubicCapacity**, κλπ. καθώς και ***μεθόδους, όπως*** για παράδειγμα, **void start()**, **void turnRight()**, **void stop()**, **void turnLightsOn()**, κλπ.

Κλάσεις (3)

- Όπως η Java έχει ορίσει και παρέχει κλάσει όπως String, BigInteger, StringBuilder, Scanner, κλπ. έτσι και εμείς μπορούμε να ορίσουμε δικές μας, **User-Defined κλάσεις**
- Ο ορισμός μιας κλάσης **επέχει θέση Τύπου Δεδομένων**, μπορούμε δηλαδή στη συνέχεια να ορίζουμε μεταβλητές αυτού του Τύπου Δεδομένων. **Για αυτό αναφερόμαστε στις κλάσεις και ως Τύποι (Types)**
- Οι **μεταβλητές κλάσεων ονομάζονται αντικείμενα (objects) ή στιγμιότυπα (instances)** και τεχνικά είναι αναφορές (references) προς τα περιεχόμενα του instance, αφού πρόκειται για σύνθετο τύπο

Αντικειμενοστραφείς γλώσσες

- Όλες οι νεότερες γλώσσες προγραμματισμού όπως η **Java,** C++, C#, PHP, Python, JavaScript κλπ. παρέχουν τον τύπο δεδομένων ***Κλάση (Class)*** που ***ενθυλακώνει πεδία και μεθόδους*** ώστε να μπορεί να αναπαριστά αποτελεσματικά τον πραγματικό κόσμο

- Μιας και το βασικό δομικό στοιχείο των γλωσσών αυτών είναι οι Κλάσεις και οι μεταβλητές των Κλάσεων, δηλαδή τα στιγμιότυπα ή ***αντικείμενα (objects)***, οι γλώσσες αυτές ονομάζονται **Αντικειμενοστραφείς γλώσσες προγραμματισμού (Object-Oriented Programming Languages)**

Συναρτησιακές γλώσσες Προγραμματισμού

- Ιστορικά δεν υπήρχαν μόνο Αντικειμενοστραφείς γλώσσες. Αρχικά υπήρχαν Συναρτησιακές ή Διαδικασιακές (Functional/Procedural) γλώσσες προγραμματισμού , όπως η C, Pascal που δεν περιείχαν Κλάσεις, αλλά μόνο συναρτήσεις/διαδικασίες
- Στον Αντικειμενοστραφή Προγραμματισμό, οι μέθοδοι, όπως έχουμε αναφέρει, είναι ουσιαστικά συναρτήσεις που όμως είναι ενσωματωμένες μέσα σε κλάσεις. Στο σημείο αυτό έγκειται η διαφορά στην διαφορετική ορολογία Συνάρτηση/Μέθοδος
- Οι συναρτήσεις των Συναρτησιακών Γλωσσών, μπορούν να δηλώνονται ως ανεξάρτητα μπλοκ κώδικα, όχι μέσα σε Κλάσεις, οι οποίες όπως αναφέραμε, στις γλώσσες όπως C και Pascal δεν υπάρχουν

Μικτές Γλώσσες Προγραμματισμού

- Σε γλώσσες όπως η JavaScript, Python δίνεται η δυνατότητα ορισμού Κλάσεων όπου μπορούμε και δηλώνουμε μεθόδους, αλλά ταυτόχρονα μπορούμε και δηλώνουμε και ανεξάρτητες συναρτήσεις
- Επομένως οι **JavaScript**, **Python** είναι και Συναρτησιακές αλλά και Αντικειμενοστραφείς γλώσσες προγραμματισμού

Java Lambdas

- Επειδή οι ανεξάρτητες συναρτήσεις είναι ευέλικτος και σύντομος τρόπος προγραμματισμού, από την Java 8 και μετά, προστέθηκαν στην Java τα **Lambda Expressions** (ή απλά **Lambdas**) τα οποία συμπεριφέρονται σαν συναρτήσεις (είναι συναρτήσεις) και τα οποία θα δούμε αργότερα στα μαθήματα που ακολουθούν

Αντικειμενοστραφής Προγραμματισμός

- Προγραμματιστικά, ο Αντικειμενοστραφής Προγραμματισμός, δεν αφορά μόνο τον τύπο δεδομένων Class, αλλά με βάση τον τύπο δεδομένων Class καθώς και τα αντικείμενα/στιγμιότυπα των κλάσεων, υπάρχει μία γενικότερη προγραμματιστική προσέγγιση: **ο Αντικειμενοστραφής ή Αντικειμενοστρεφής Προγραμματισμός (Object-Oriented Programming)**
- Ας δούμε συνοπτικά, ένα outline κάποιων σημαντικών χαρακτηριστικών του Αντικειμενοστραφούς Προγραμματισμού πριν τα δούμε αναλυτικά

Αντικειμενοστραφής Προγραμματισμός (1)

- Για να έχουμε τη μεγάλη εικόνα πριν ξεκινήσουμε να βλέπουμε τα επιμέρους θέματα, να σημειώσουμε ότι τα θέματα που αποτελούν τον Αντικειμενοστραφή Προγραμματισμό είναι τα εξής:
- **Ορισμός** Κλάσεων και δηλώσεις αντικειμένων κλάσεων
- **Ενθυλάκωση** (Encapsulation), δηλαδή η δυνατότητα των κλάσεων να αποκρύπτουν τα πεδία τους και τις μεθόδους τους από άλλες κλάσεις

Αντικειμενοστραφής Προγραμματισμός (2)

- **Μέθοδοι** κλάσεων, Λειτουργικότητα
- **Κληρονομικότητα** (Inheritance), δηλαδή η επαναχρησιμοποίηση ήδη υπαρχουσών κλάσεων ώστε να δημιουργήσουμε νέες πιο ειδικές κλάσεις
- **Υπερκάλυψη** (Override) – Virtual Methods, δηλαδή η δυνατότητα να κληρονομήσουμε μεθόδους και να αλλάξουμε το σώμα τους, διατηρώντας σταθερή την υπογραφή τους

Αντικειμενοστραφής Προγραμματισμός (3)

- **Πολυμορφισμός** Polymorphism, δηλαδή να μπορούμε σε δηλώσεις κλάσεων (ή interfaces) που είναι στην κορυφή μίας ιεραρχίας κληρονομικότητας να χρησιμοποιούμε ως πραγματικές μεταβλητές αντικείμενα που βρίσκονται σε οποιοδήποτε σημείο της ιεραρχίας κληρονομικότητας (ή υλοποιούν το interface)

Αντικειμενοστραφής Προγραμματισμός (4)

- **Abstract κλάσεις**, δηλαδή κλάσεις οι οποίες μπορεί να περιέχουν abstract μεθόδους, δηλαδή μεθόδους χωρίς σώμα, αλλά μόνο την επικεφαλίδα και τις οποίες να κάνουν override οι κλάσεις που κληρονομούν από τις abstract κλάσεις

Αντικειμενοστραφής Προγραμματισμός (5)

- **Ανώνυμες κλάσεις**, δηλαδή κλάσεις που δημιουργούνται απευθείας με new
- **Ειδικές κλάσεις της Java**, όπως η κλάση *Object* από την οποία κληρονομούν έμμεσα όλες οι κλάσεις στην Java (της ίδιας της Java αλλά και δικές μας proprietary κλάσεις)

Αντικειμενοστραφής Προγραμματισμός (6)

- **Interfaces**, δηλαδή γκρουπ από abstract μεθόδους τις οποίες πρέπει να υλοποιήσουν οι κλάσεις (ή οι abstract κλάσεις) που κληρονομούν από το interface (ή καλύτερα *υλοποιούν* το interface)

Αντικειμενοστραφής Προγραμματισμός (7)

- **Ειδικά interfaces**, όπως *Functional interfaces* (τα οποία περιέχουν μία μόνο μέθοδο), *Marker interfaces* (τα οποία δεν περιέχουν καμία μέθοδο)
- **Ειδικά interfaces της Java** (που τελειώνουν σε –able), όπως *Cloneable, Runnable, Comparable, Serializable* και άλλα που δίνουν τη δυνατότητα στις κλάσεις που τα υλοποιούν να κάνουν κάτι (-able)

Αντικειμενοστραφής Προγραμματισμός (8)

- **Lambda Expressions**, που δίνουν τη δυνατότητα να αναπαραστήσουμε Functional interfaces με lambdas ώστε να μειώσουμε τον κώδικα και την πολυπλοκότητα που έχουν οι ανώνυμες κλάσεις όταν πρόκειται για μία μόνο μέθοδο και να μπορούμε επίσης να χρησιμοποιούμε callbacks, μετατρέποντας πολλές φορές ένα εγγενή επαναληπτικό μηχανισμό (όπως στην επεξεργασία collections) σε functional

Αντικειμενοστραφής Προγραμματισμός (9)

- **Ειδικά Functional Interfaces**, που παρέχει η Java, όπως Consumer, Predicate, Function, κλπ. που διευκολύνουν να περνάμε callbacks (μεθόδους ως παραμέτρους άλλων μεθόδων) σε μορφή Lambda χωρίς να χρειάζεται να γράφουμε εμείς κώδικα

Αντικειμενοστραφής Προγραμματισμός (9)

- **Java Collections**, που είναι έτοιμες δομές δεδομένων, όπως **List**, ArrayList, LinkedList, **Set**, HashSet, **Map**, HashMap, **Queue**, Deque, κλπ. που μας διευκολύνουν να χρησιμοποιούμε έτοιμες δομές δεδομένων χωρίς να χρειάζεται να τις ορίζουμε εμείς

Χαρακτηριστικά Ο-Ο Programming

- Τα προηγούμενα ήταν ένας συνοπτικός οδηγός του τι περιλαμβάνεται στον Αντικειμενοστραφή Προγραμματισμό και τα οποία θα δούμε στη συνέχεια

Κατανεμημένος Προγραμματισμός (1)

- Μέχρι στιγμής έχουμε δει τις κλάσεις να περιλαμβάνουν κυρίως μεθόδους καθώς και τη μέθοδο main από την οποία ξεκινά και εκτελείται κάθε πρόγραμμα στην Java
- Κατά αυτό τον τρόπο είχαμε **συνδέσει (couple)** μέσα σε μια κλάση τόσο το μέρος της εκτέλεσης ενός προγράμματος (την main), όσο και τη λογική των μεθόδων που χρησιμοποιούσε η main

Κατανεμημένος Προγραμματισμός (2)

- Τεχνικά αυτό είναι επιτρεπτό, και ο λόγος που το κάναμε ήταν γιατί δώσαμε έμφαση στο προγραμματιστικό μέρος του Δομημένου προγραμματισμού
- Μεθοδολογικά ωστόσο, το να είναι στενά συνδεδεμένες (tightly coupled) μέσα σε μία κλάση τόσο η main όσο και οι άλλες μέθοδοι της κλάσης δεν είναι επαναχρησιμοποιήσιμο (reusable)

Κατανεμημένος Προγραμματισμός (3)

- Ιδανικά θα θέλαμε να βλέπουμε τις μεθόδους ή/και τα δεδομένα που χρησιμοποιεί η main ως ανεξάρτητες οντότητες κώδικα που βρίσκονται μέσα σε ανεξάρτητες κλάσεις και η main απλά να είναι μόνη της σε μία άλλη κλάση και να χρησιμοποιεί τις μεθόδους ανεξάρτητων κλάσεων
- Θα θέλαμε δηλαδή να κάνουμε *decouple*, να αποσυνδέσουμε την main, που είναι μια ειδική μέθοδος από τη λογική της εφαρμογής μας

Κατανεμημένος Προγραμματισμός (4)

- Κατά αυτό τον τρόπο, αν έχουμε ανεξάρτητες κλάσεις, χωρίς να περιέχουν κάποια main, θα είναι reusable γιατί θα μπορούν να χρησιμοποιούνται από οποιαδήποτε main
- Θα έχουμε δηλαδή μία λογική μίας ανεξάρτητης κλάσης, που είναι πάροχος υπηρεσιών (μεθόδων) και κάποιων άλλων κλάσεων (όπως για παράδειγμα της κλάσης που θα περιέχει την main) που θα «αιτούνται» των υπηρεσιών των ανεξάρτητων κλάσεων

Κατανεμημένος Προγραμματισμός (5)

- Θα έχουμε δηλαδή μία λογική διάρθρωση/αρχιτεκτονική ενός μοντέλου που ονομάζεται client-server (πελάτης-εξυπηρετητής), όπου η ανεξάρτητη κλάση παρέχει υπηρεσίες/μεθόδους και η κλάση πελάτης τις καλεί και τις χρησιμοποιεί
- Υπάρχει επομένως λογικός διαχωρισμός. Θα μπορούσε να υπάρχει και φυσικός διαχωρισμός αν ο πελάτης και ο εξυπηρετητής ήταν σε διαφορετικά μηχανήματα

Εμβέλεια / Scoping

- Πριν περάσουμε στον Αντικειμενοστραφή Προγραμματισμό, θα δούμε ξανά συνοπτικά το θέμα της εμβέλειας των μεταβλητών
- Στο επίπεδο των μεθόδων, **μεταβλητές μπορούμε να δηλώνουμε σε τρία σημεία**
  - Στις **τυπικές παραμέτρους μίας μεθόδου**
  - Στο **σώμα της μεθόδου**
  - Στις **δομές ελέγχου (if, while, for, switch/case)** που πιθανά υπάρχουν μέσα στο σώμα της μεθόδου

Εμβέλεια / Local Scoping (1)


```java
package testbed.ch10;

public class Scoping {

    public static void main(String[] args) {
        int result;
        int num1 = 1;
        int num2 = 2;

        result = add(num1, num2);
        System.out.println(result);
    }

    public static int add(int a, int b) {
        int result;
        result = a + b;
        return result;
    }
}
```

- Έχουμε δει την έννοια της εμβέλειας (scoping) μίας μεταβλητής στο πλαίσιο των μεθόδων
- Αν έχουμε μία μέθοδο **add**, τότε το result, (αλλά και τα a, b) είναι **τοπικές (local) μεταβλητές** της μεθόδου add. Αυτό σημαίνει πως έξω από την add δεν υπάρχουν αυτές οι μεταβλητές.
- Το ίδιο ισχύει και για το result και num1, num2 της **main**. Σημειωτέον, είναι άλλο το result της add και άλλο το result της main

Εμβέλεια / Local Scoping (2)


```
1   package testbed.ch10;
2
3   public class Scoping {
4
5       public static void main(String[] args) {
6           int result;
7           int num1 = 1;
8           int num2 = 2;
9
10          result = add(num1, num2);
11          System.out.println(result);
12      }
13
14      public static int add(int a, int b) {
15          int result;
16          result = a + b;
17          return result;
18      }
19  }
```

- Αυτός ο τύπος διαχείρισης εμβέλειας ονομάζεται **lexical scoping** γιατί μπορούμε να συνάγουμε την εμβέλεια μίας μεταβλητής απλά από την ανάγνωση κώδικα
- Τα curly brackets { } των μεθόδων ορίζουν χώρους τοπικής εμβέλειας (Local Scoping)

Scoping / Block Level


```
public static int div(int a, int b) {
    int result = 0;

    if (b == 0) {
        boolean isZero = true;
        System.out.println("is zero: " + isZero);
        return result;
    }

    return a / b;
}
```

Γενικά, στην περίπτωση που έχουμε nested scopes, οι μεταβλητές των outer scopes έχουν πρόσβαση στα inner scopes, όχι όμως και το αντίθετο

- Άλλο σημείο όπου μπορούμε να κάνουμε δηλώσεις μεταβλητών είναι μέσα σε δομές ελέγχου όπως if, while, for (**block scoping**)
- Στο παράδειγμα η εμβέλεια του **isZero** είναι μόνο μέσα στην if. Έξω από την if το isZero δεν υπάρχει, δεν μπορούμε να αναφερθούμε σε αυτό.
- Ωστόσο, το **result** που είναι τοπική μεταβλητή της div έχει εμβέλεια σε όλο το scope της div, άρα και μέσα στην if. Το scope της div είναι το outer scope, ενώ το scope της if είναι το inner scope

Class Scoping


```
1  package testbed.ch10;
2
3  public class Scoping {
4      static int result = 20;
5
6      public static void main(String[] args) {
7          int result;
8          int num1 = 1;
9          int num2 = 2;
10
11         result = add(num1, num2);
12
13         switch (result) {
14             case 1:
15                 int i = 0;
16
17         }
18         System.out.println(result);
19     }
```

- Στην περίπτωση που δηλώνουμε σε επίπεδο κλάσης, έξω από όλες τις μεθόδους, τότε οι μεταβλητές έχουν εμβέλεια σε όλες τις μεθόδους της κλάσης
- Αν μέσα σε κάποια μέθοδο, έχουμε δήλωση μεταβλητής με το ίδιο όνομα, υπερισχύει το local scoping

Απλές μορφές κλάσεων

- Πριν πάμε σε πιο σύνθετες μορφές κλάσεων θα δούμε **μία απλή μορφή κλάσης που περιέχει μόνο πεδία (fields) δηλαδή αναπαριστά μόνο δεδομένα (data class)**
- Μια κλάση μπορεί να περιέχει πεδία διαφόρων τύπων
- Έστω η κλάση ***Student*** **που περιέχει τον κωδικό του Student, το επίθετο και το όνομα**

Απλή κλάση Student


```java
package gr.aueb.cf.testbed;

/**
 * Ορισμός μιας απλής κλάσης Student.
 */
public class Student {
    int id;
    String firstname;
    String lastname;
}
```

- Η κλάση Student περιέχει τρία πεδία (fields), διαφόρων απλών και σύνθετων τύπων (int και String)
- Ωστόσο δεν έχουμε χαρακτηρισμό ορατότητας (visibility/scoping) για τα πεδία, αν δηλαδή τα πεδία είναι public, private κλπ. (βλ. επόμενη διαφάνεια) κάτι που είναι πολύ σημαντικό γιατί καθορίζει το πεδίο εμβέλειας των πεδίων ή με άλλα λόγια το που θα είναι γνωστά αυτά πεδία δηλαδή το ποιος θα έχει πρόσβαση σε αυτά τα πεδία.

Χαρακτηρισμοί πρόσβασης (Modifiers)

- Στο επίπεδο της κλάσης, η εμβέλεια των πεδίων της κλάσης, ορίζεται από τον τρόπο χαρακτηρισμού (**access modifiers**) του κάθε πεδίου της κλάσης
- Τα **πεδία μίας κλάσης** μπορούν να χαρακτηριστούν ως:
  - **Δημόσια (public)**, που είναι προσπελάσιμα από παντού, δηλαδή από όλες τις κλάσεις του ίδιου ή άλλων packages ή άλλων projects
  - **Ιδιωτικά (private)** που είναι προσπελάσιμα μόνο μέσα από την ίδια την κλάση στην οποία ορίζονται
  - **Προστατευμένα (protected)** που είναι προσπελάσιμα από την ίδια την κλάση και από τις υποκλάσεις σε μια ιεραρχία κληρονομικότητας καθώς από τις κλάσεις του ίδιου package
  - **Χωρίς χαρακτηρισμό (default / package private)** που είναι προσπελάσιμα από την ίδια την κλάση και μόνο από τις κλάσεις του package, όπως στο παράδειγμα Student

Αντικείμενα κλάσεων


```java
package gr.aueb.cf.testbed;

/**
 * Ορισμός μιας απλής κλάσης Student.
 */
public class Student {
    int id;
    String firstname;
    String lastname;
}
```

```java
package gr.aueb.cf.testbed;

/**
 * Creates a Student instance named alice.
 */
public class StudentDriver {

    public static void main(String[] args) {
        Student alice = new Student();
    }
}
```

- Η κλάση ***Student* επέχει θέση τύπου δεδομένων** και αφού οριστεί (μέσα σε ένα αρχείο Student.java), μπορούμε στη συνέχεια να ορίζουμε μεταβλητές αυτής της κλάσης μέσα σε άλλες κλάσεις, όπως σε μία κλάση που περιέχει μόνο την main
- Για παράδειγμα μπορούμε να δηλώσουμε στην main ένα αντικείμενο / instance με όνομα ***alice* τύπου *Student***
- Οι μεταβλητές κλάσεων ονομάζονται ***αντικείμενα ή στιγμιότυπα (objects or instances)*** της κλάσης

Δημιουργία αντικειμένου


```java
package gr.aueb.cf.testbed;

/**
 * Creates a Student instance named alice.
 */
public class StudentDriver {

    public static void main(String[] args) {
        Student alice = new Student();
    }
}
```

- Η ***δήλωση*** του ***alice*** γίνεται όπως η δήλωση όλων των μεταβλητών ενώ η ***δημιουργία*** του instance με όνομα ***alice*** γίνεται με την *new* η οποία καλεί την μέθοδο `Student()` που ονομάζεται ***κατασκευαστής*** *ή* ***δημιουργός (constructor)*** *και* ***έχει το ίδιο όνομα με την κλάση*** (παρατηρήστε ότι έχει παρενθέσεις, άρα είναι μέθοδος)
- Ο ***constructor*** όταν δεν έχει τυπικές παραμέτρους (non-argument constructor) όπως ο `Student()` ονομάζεται ***default constructor*** και ***παρέχεται αυτόματα από την Java***

Κατασκευαστές / Δημιουργοί / Constructors

- Οι constructors **έχουν το ίδιο όνομα με την κλάση.** Σκοπός των δημιουργών είναι: να **αρχικοποιούν τα πεδία** του instance μιας κλάσης σε συγκεκριμένες τιμές (state)
- Ο **default constructor** δεν παίρνει παραμέτρους και αρχικοποιεί τα πεδία σε default τιμές, δηλαδή τα Strings σε null, τα int σε 0, float/double σε 0.0 και boolean σε false (στην πραγματικότητα σε αυτή την περίπτωση το JVM είναι που αρχικοποιεί)

Δημιουργία αντικειμένου (1)

- Όπως με τις σύνθετες δομές δεδομένων String και πίνακες, έτσι και με τις κλάσεις, η δήλωση:

```
5      public static void main(String[] args) {
6          Student alice = new Student();
```

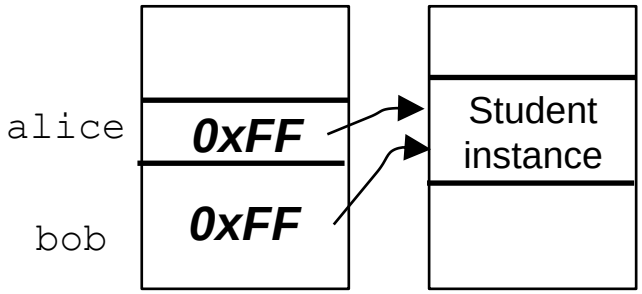
κάνει δύο πράγματα:

  - ***Student alice*** -- δηλώνει την αναφορική μεταβλητή ***alice*** και κατανέμει χώρο για αυτήν στο stack της μνήμης. Αυτό γίνεται κατά το χρόνο μεταγλώττισης (στατική δέσμευση)
  - ***= new Student();*** -- όπου κατανέμει χώρο στο heap για το πραγματικό αντικείμενο (instance) και επιστρέφει ένα δείκτη (διεύθυνση μνήμης) στο ***alice***. Το new γίνεται κατά το χρόνο εκτέλεσης (δυναμική δέσμευση)

Δημιουργία αντικειμένου (2)


```
5      public static void main(String[] args) {
6          Student alice = new Student();
```

- Λέμε ότι η αναφορική μεταβλητή ***alice*** δείχνει στο πραγματικό αντικείμενο/στιγμιότυπο (instance), που έχει δημιουργηθεί με τη new
- Αν στη συνέχεια δηλώσω ***Student bob = alice;*** Τότε και το bob θα δείχνει στην ίδια θέση που δείχνει και η alice
- Άρα και εδώ θα πρέπει να εξετάσουμε θέματα που έχουν να κάνουμε immutable / mutable κλάσεις



Public modifier


```java
package testbed.ch10;

/**
 * Defines a simple type with
 * public access modifiers.
 */
public class Student {
    public int id;
    public String firstname;
    public String lastname;
}
```

- Όπως αναφέραμε ο χαρακτηρισμός των πεδίων ως **public** είναι χαρακτηρισμός ορατότητας (visibility)
- Στην πραγματικότητα είναι χαρακτηρισμός πρόσβασης (access) στα πεδία της κλάσης μας από την ίδια ή από άλλες κλάσεις

Τα πεδία που είναι public **είναι άμεσα προσπελάσιμα από οποιαδήποτε άλλη κλάση**, στο ίδιο package, σε άλλα packages του ίδιου project, καθώς και σε άλλα projects, από όλο δηλαδή το Java world!

Access σε public fields


```java
package testbed.ch10;

/**
 * Creates an instance (alice) of type
 * Student and has access to alice's
 * state, that is, alice's fields
 *
 */
public class StudentApp {

    public static void main(String[] args) {
        Student alice = new Student();

        alice.id = 1;
        alice.firstname = "Alice";
        alice.lastname = "W.";

        System.out.println("Id: " + alice.id);
        System.out.println("Firstname" + alice.firstname);
        System.out.println("Lastname" + alice.lastname);
    }
}
```

- Έστω ένα instance με όνομα alice, που δημιουργεί η main
- Η main() με τον **τελεστή τελεία (dot notation)** έχει άμεση πρόσβαση στα *public* πεδία ενός instance, έχει άμεση πρόσβαση στο state* του instance και μπορεί να το αλλάξει
- * Οι συγκεκριμένες τιμές των πεδίων ενός instance ονομάζεται: το state του instance

State του Instance

- Αν οι access modifiers των πεδίων μίας κλάσης είναι public, τότε, όπως είδαμε μπορεί μία άλλη κλάση να έχει πρόσβαση στα πεδία της κλάσης
- Είναι ασφαλές αυτό;
- Να μπορεί ο οποιοσδήποτε τρίτος να έχει πρόσβαση και να αλλάζει το state ενός αντικειμένου;

State και Consistency

- Όταν θέτουμε τα πεδία μιας κλάσης σε public προκύπτουν προβλήματα ασφάλειας, γιατί ιδανικά δεν θέλουμε κανείς να έχει άμεση πρόσβαση στα πεδία των instances μιας κλάσης.
- Οι **τιμές των πεδίων** ενός instance συνιστούν το **state** του instance και θέλουμε το state των instances να είναι **συνεπές**
- Να μην μπορεί δηλαδή κανείς να έχει άμεση πρόσβαση σε αυτό εκτός από την ίδια την κλάση/instance

Ενθυλάκωση / Encapsulation

- Μία βασική ιδιότητα στον αντικειμενοστραφή προγραμματισμό είναι η ***ενθυλάκωση*** (encapsulation)
- Η ***ενθυλάκωση*** αφορά:
  - Τον χαρακτηρισμό των πεδίων μίας κλάσης ως ***private*** ώστε να **αποκρύβουμε τα μέλη της κλάσης από τον «έξω κόσμο»** δηλαδή από άλλες κλάσεις μέσα ή έξω από το package μας

Βασική αρχή ενθυλάκωσης

Τα πεδία μιας κλάσης θα πρέπει να ΄αποκρύπτονται΄ από τον ΄έξω κόσμο΄, δηλαδή να είναι private

- Ο *΄έξω κόσμος΄* είναι όλες οι άλλες κλάσεις εκτός από την κλάση μέσα στην οποία ορίζουμε τα πεδία
- Αν τα πεδία μιας κλάσης δεν είναι private, ***εκθέτουμε την εσωτερική υλοποίηση της κλάσης*** και μπορεί να παραβιαστεί η ασφάλεια και η ακεραιότητα των δεδομένων των instances της κλάσης αφού άλλες κλάσεις θα μπορούν να έχουν άμεση πρόσβαση στα δεδομένα της κλάσης καθώς και να ΄βλέπουν΄ την εσωτερική υλοποίηση της κλάσης

Method encapsulation

- Η ενθυλάκωση (encapsulation) επεκτείνεται και στις μεθόδους μιας κλάσης
- Μέθοδοι που αφορούν την εσωτερική υλοποίηση (internals) των λειτουργιών μιας κλάσης πρέπει να είναι private

Κλάση Student


```java
package gr.aueb.cf.testbed;

/**
 * Ορισμός μιας κλάσης Student
 * με private πεδία (encapsulation).
 */
public class Student {
    private int id;
    private String firstname;
    private String lastname;
}
```

- Ορίζουμε μία κλάση Student, όπου τα πεδία id, firstname, lastname είναι **private**

Πρόσβαση σε private fields


```
1  package gr.aueb.cf.testbed;
2
3  /**
4   * Instantiates a Student (alice)
5   * and sets values to instance's fields.
6   */
7  public class StudentDriver {
8
9      public static void main(String[] args) {
10         Student alice = new Student();
11
12         alice.id = 1;
13         alice.f
14         alice.l
15     }
16 }
17
```

'id' has private access in 'gr.aueb.cf.testbed.Student'

Create field 'id' in 'Student' Alt+Shift+Enter More actions... Alt+Enter

```
gr.aueb.cf.testbed.Student
private int id
```

- Παρατηρούμε ότι από την main() δεν μεταγλωττίζεται η πρόσβαση με την τελεία όταν τα πεδία του instance είναι private
- ***Πως μπορούμε όμως τότε να έχουμε πρόσβαση στα πεδία***, αφού σε private πεδία δεν μπορούμε να χρησιμοποιήσουμε τον τελεστή τελεία;

Setters & Getters (Mutators & Accessors)

- Για να παρέχει μία κλάση πρόσβαση στα private πεδία της, ορίζει ειδικές public μεθόδους
  - ***Setters (****Mutators)*, που είναι μέθοδοι που εκχωρούν (set) τιμές στα πεδία της κλάσης και κατά σύμβαση ξεκινούν με τη λέξη ***set*** (π.χ. setFirstname).
  - ***Getters (****Accessors)*, που είναι μέθοδοι που κάνουν return την τιμές των πεδίων της κλάσης και κατά σύμβαση ξεκινούν με τη λέξη ***get*** (π.χ. getFirstname)
- Έστω πεδίο με όνομα x. Οι συμβάσεις ονοματοδοσίας για τους getters και setters είναι **getX και setX** αντίστοιχα. Για boolean πεδία, εκτός από getX, επιτρέπεται και η σύμβαση **isX**

Κλάση Student


```java
package gr.aueb.cf.testbed;

/**
 * Student class.
 */
public class Student {
    private int id;
    private String firstname;
    private String lastname;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }
}
```

- Πρόκειται για μια απλή κλάση Java με private πεδία και μεθόδους πρόσβασης (setters / getters)
- Για κάθε private πεδίο έχουμε ένα setter και ένα getter, που είναι public

Driver class για κλάση Student


```java
package gr.aueb.cf.testbed;

/**
 * Instantiates a Student (alice) and uses setters and getters
 * to set / get values to / from instance's fields.
 */
public class StudentDriver {

    public static void main(String[] args) {
        Student alice = new Student();

        alice.setId(1);
        alice.setFirstname("Alice");
        alice.setLastname("Wonderland");

        System.out.println("ID: " + alice.getId());
        System.out.println("First Name: " + alice.getFirstname());
        System.out.println("Last Name: " + alice.getLastname());
    }
}
```

- Σε μια άλλη κλάση έχουμε μια main() όπου δημιουργούμε ένα αντικείμενο ***alice*** τύπου ***Student*** και καλούμε τους setters / getters για εκχώρηση / ανάκληση τιμών αντίστοιχα

- Με τη χρήση των setters εκχωρούμε τιμές στα πεδία του αντικειμένου και με τους getters διαβάζουμε/ανακαλούμε αντίστοιχα τις τιμές των πεδίων μέσα στην *println()*

Default Constructor (1)


```
1  package testbed.ch10;
2
3  /**
4   * Student class.
5   */
6  public class Student {
7      public int id;
8      public String firstname;
9      public String lastname;
10
11     public Student() {
12         id = 0;
13         firstname = null;
14         lastname = null;
15     }
16
17     public int getId() { return id; }
20     public void setId(int id) { this.id = id; }
23     public String getFirstname() { return firstname; }
26     public void setFirstname(String firstname) { this.firstname = firstname; }
29     public String getLastname() { return lastname; }
32     public void setLastname(String lastname) { this.lastname = lastname; }
35 }
```

- Στις γραμμές 11-15 **παρέχεται και ένας constructor**, δηλαδή **μία ειδική μέθοδος που έχει ίδιο όνομα με την κλάση**
- Σκοπός των constructors είναι να αρχικοποιήσουν τα πεδία της κλάσης ή πιο σωστά να αρχικοποιήσουν ένα instance μίας κλάσης σε ένα αρχικό state
- Ο συγκεκριμένος constructor δεν έχει παραμέτρους και αρχικοποιεί τα πεδία σε default τιμές. Ονομάζεται **default constructor** ή argumentless constructor

Default Constructor (2)


```
1  package gr.aueb.cf.testbed;
2
3  /**
4   * Student class.
5   */
6  public class Student {
7      private int id;
8      private String firstname;
9      private String lastname;
10
11     // Default Constructor
12     public Student() {
13     }
14
15     public int getId() {
16         return id;
17     }
18
19     public void setId(int id) {
20         this.id = id;
21     }
22
23     public String getFirstname() {
24         return firstname;
25     }
```

```
26
27     public void setFirstname(String firstname) {
28         this.firstname = firstname;
29     }
30
31     public String getLastname() {
32         return lastname;
33     }
34
35     public void setLastname(String lastname) {
36         this.lastname = lastname;
37     }
38 }
```

- Εναλλακτικά μπορεί ο default constructor (γραμμές 12-13) να μην έχει σώμα, αφού το JVM όταν δεν δίνουμε εμείς τιμές στα πεδία της κλάσης τα αρχικοποιεί με default τιμές

Default Constructor (3)


```java
package gr.aueb.cf.testbed;

/**
 * Student class.
 */
public class Student {
    private int id;
    private String firstname;
    private String lastname;

    // Default Constructor
    public Student() {
    }
```

- Οι constructors είναι επομένως **μέθοδοι που έχουν το ίδιο όνομα με την κλάση και σκοπός τους είναι να αρχικοποιήσουν τα πεδία της κλάσης**
- Ο **Default Constructor** είναι ένας constructor χωρίς παραμέτρους (parameter-less) που αρχικοποιεί τα πεδία της κλάσης σε default τιμές
- Στη συγκεκριμένη περίπτωση αρχικοποιεί το id στο 0 και τα δύο Strings στο null

Default Constructor (4)


```java
package testbed;

/**
 * Student Class.
 */
public class Student {
    private int id;
    private String firstname;
    private String lastname;

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getFirstname() { return firstname; }
    public void setFirstname(String firstname) { this.firstname = firstname; }
    public String getLastname() { return lastname; }
    public void setLastname(String lastname) { this.lastname = lastname; }
}
```

- Αν δεν ορίσουμε ρητά Default Constructor τότε η Java (JVM) μας παρέχει έμμεσα ένα Default Constructor

POJOs και Java beans

- Οι απλές κλάσεις στην Java (που δεν σχετίζονται με κάποιο framework) αναφέρονται και ως POJOs (Plain Old Java Objects). Τα POJOs δεν εξαρτώνται από 3rd party frameworks, δεν χρησιμοποιούν conventions για τους setters/getters, ούτε έχουν αναγκαστικά πεδία και constructors
- Σε **POJOs** που επιπλέον έχουν ένα default constructor και getters/setters με conventions (getX, isX, setX) θα αναφερόμαστε ως **Java Beans** (στο παρόν θα χρησιμοποιούμε ένα loose definition του JavaBean. Στα επίσημα specs JDK 1.1 τα JavaBeans πρέπει να είναι και Serializable)

POJO – Java Bean


```java
package gr.aueb.cf.testbed;

/**
 * Student POJO class. Java Bean.
 */
public class Student {
    private int id;
    private String firstname;
    private String lastname;

    // Default Constructor
    public Student() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstname() {
        return firstname;
    }

    public void setFirstname(String firstname) {
        this.firstname = firstname;
    }

    public String getLastname() {
        return lastname;
    }

    public void setLastname(String lastname) {
        this.lastname = lastname;
    }
}
```

- Ένα **Java Bean** είναι απλά μία data class με seters/getters και ένα default constructor

Υπερφορτωμένοι constructors

- Εκτός από τον default constructor που παρέχει αυτόματα από το JVM, μπορεί να θέλουμε να έχουμε και άλλους constructors με λίστα παραμέτρων
- Για παράδειγμα μπορεί να θέλουμε για ευκολία (convenience) αρχικοποίησης των πεδίων της κλάσης Student σε τιμές που θέλουμε, να ορίσουμε τον constructor:

public Student(int id, String firstname, String lastname)

- Ο constructor αυτός ονομάζεται υπερφορτωμένος (**overloaded**) μιας και έχει το ίδιο όνομα με τον default (όλοι οι constructors έχουν το ίδιο όνομα) αλλά διαφορετική λίστα παραμέτρων, δηλαδή διαφορετική υπογραφή (signature)

Υπερφόρτωση

- Όπως έχουμε πει, η ιδιότητα της υπερφόρτωσης (overload) μεθόδων στην Java μας επιτρέπει να ορίζουμε μεθόδους με το ίδιο όνομα αλλά να διαφοροποιούνται οι τυπικές παράμετροι, δηλαδή διαφορετική υπογραφή
- Ιδιαίτερα στους constructors -**που πρέπει όλοι να έχουν το ίδιο όνομα με την κλάση**- όταν ορίζουμε και άλλους constructors που έχουν παραμέτρους λέμε πως είναι υπερφορτωμένοι

Constructors

- Όταν ορίσουμε έναν ή περισσότερους υπερφορτωμένους constructors η Java **δεν μας παρέχει αυτόματα default constructor, οπότε θα πρέπει (αν τον θέλουμε) να τον ορίσουμε ρητά**

Overloaded Constructor


```java
package gr.aueb.cf.testbed;

/**
 * Student POJO class. Java Bean.
 */
public class Student {
    private int id;
    private String firstname;
    private String lastname;

    // Default Constructor
    public Student() {
    }

    // Overloaded Constructor
    public Student(int id, String firstname, String lastname) {
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
    }
```

- Ο overloaded constructor μας δίνει τη δυνατότητα να μπορούμε να αρχικοποιούμε ένα instance σε οποιοδήποτε state
- Παρατηρούμε το ***this*** στο σώμα του overloaded constructor. ***Το this*** είναι μία προκαθορισμένη αναφορά (δείκτης) της Java που δείχνει στο ***ίδιο το αντικείμενο*** (τρέχον, όποιο κι αν είναι αυτό)
- Με το **this.id, this.firstname, this.lastname** εννοούμε τα πεδία της κλάσης, ενώ με το **id, firstname, lastname** εννοούμε τις τυπικές παραμέτρους

Driver Class


```
1  package gr.aueb.cf.testbed;
2
3  /**
4   * Instantiates two Students (alice & Bob) with
5   * Default and Overloaded Constructors, respectively.
6   */
7  public class StudentDriver {
8
9      public static void main(String[] args) {
10         Student alice = new Student();
11         Student bob = new Student(2, "Bob", "Marley");
12
13         alice.setId(1);
14         alice.setFirstname("Alice");
15         alice.setLastname("Wonderland");
16
17         System.out.println("ID: " + alice.getId());
18         System.out.println("First Name: " + alice.getFirstname());
19         System.out.println("Last Name: " + alice.getLastname());
20
21         System.out.println("ID: " + bob.getId());
22         System.out.println("First Name: " + bob.getFirstname());
23         System.out.println("Last Name: " + bob.getLastname());
24     }
25 }
```

- Στη γραμμή 10 καλούμε τον default constructor
- Στην γραμμή 11 καλούμε τον - ***overloaded constructor***
- Συνεπώς, η main() δημιουργεί δύο instances του τύπου Student, ένα με default τιμές (γραμμή 10) και ένα με συγκεκριμένες τιμές (γραμμή 11)
- Τυπικά η δήλωση και αρχικοποίηση αντικειμένων δεν διαφέρει από την δήλωση και αρχικοποίηση μεταβλητών σύνθετων τύπων

Overloaded Constructors


```java
package gr.aueb.cf.testbed;

/**
 * Student POJO class. Java Bean.
 */
public class Student {
    private int id;
    private String firstname;
    private String lastname;

    // Default Constructor
    public Student() {
    }

    // Overloaded Constructor
    public Student(int id, String firstname, String lastname) {
        this.id = id;
        this.firstname = firstname;
        this.lastname = lastname;
    }

    // Overloaded Constructor
    public Student(int id) {
        this.id = id;
    }
```

```java
    // Overloaded Constructor
    public Student(String firstname) {
        this.firstname = firstname;
    }

    // Overloaded Constructor
    public Student(int id, String firstname) {
        this.id = id;
        this.firstname = firstname;
    }

    // Overloaded Constructor
    public Student(String firstname, String lastname) {
        this.firstname = firstname;
        this.lastname = lastname;
    }
```

- Μπορούμε να έχουμε διάφορους συνδυασμούς overloaded constructors

Static Fields


```java
package gr.aueb.cf.testbed;

/**
 * Student static fields vs instance fields.
 */
public class Student {
    private static int studentsCount = 0;
    private int id;
    private String firstname;
    private String lastname;

    public Student() {
        studentsCount++;
    }

    public static int getStudentsCount() {
        return studentsCount;
    }

    // Getters and Setters
}
```

- Τα static fields **ανήκουν στην κλάση, όχι στα instances**. Είναι σαν global variables και διαμοιράζονται μεταξύ όλων των instances
- Αποθηκεύονται σε μία ειδική περιοχή της μνήμης του JVM που ονομάζεται Method Area (που είναι μέρος του Heap στο implementation του Oracle JVM, που ονομάζεται HotSpot JVM)
- Επομένως στο παράδειγμα, υπάρχει ένα και μόνο ένα *studentCount* που μοιράζεται μεταξύ των instances

Static Fields – Static methods


```java
package gr.aueb.cf.testbed;

/**
 * Student static fields vs instance fields.
 */
public class Student {
    private static int studentsCount = 0;
    private int id;
    private String firstname;
    private String lastname;

    public Student() {
        studentsCount++;
    }

    public static int getStudentsCount() {
        return studentsCount;
    }

    // Getters and Setters
}
```

- Στο παράδειγμα έχουμε δηλώσει ένα πεδίο *static int studentCount*, το οποίο αρχικοποιείται στο 0 και μέσα στο σώμα του Default Constructor αυξάνει κατά 1 (ακόμα και αν δεν αρχικοποιούσαμε στο 0, η default τιμή για τους int είναι 0)
- Μετράει επομένως πόσα instances έχουν δημιουργηθεί
- Έχουμε ορίσει και ένα static getter ώστε να έχουμε πρόσβαση και να διαβάζουμε το static field που είναι private
- Οι class methods (static methods) όπως και τα class fields (static fields) ανήκουν στην κλάση και όχι στα instances

Driver Class


```java
package gr.aueb.cf.testbed;

/**
 * Prints the Student-class instances' count.
 */
public class StudentDriver {

    public static void main(String[] args) {
        Student alice = new Student();

        System.out.println(Student.getStudentsCount());
    }
}
```

- Παρατηρούμε πως τις static μεθόδους όπως η *getStudentCount* τις καλούμε, όχι όπως τις instance methods, αλλά με το πρόθεμα της κλάσης δηλαδή ως *Student.getStudentsCount()*

Static block (1)


```java
package testbed.ch10;

public class StudentStaticField {
    private static int studentsCount;
    private String firstname;
    private String lastname;

    static {
        studentsCount = 0;
    }

    public StudentStaticField() {}
```

- Η αρχικοποίηση των static fields μπορεί να γίνει άμεσα στο σημείο που δηλώνονται
- Άλλος τρόπος να αρχικοποιήσουμε ένα static variable είναι μέσω ενός static block

- Τα static blocks χρησιμοποιούνται όταν έχουμε πιο περίπλοκους τρόπους αρχικοποίησης (βλ. επόμενη διαφάνεια)

Static block (2)


```java
package testbed.ch10;

import java.util.Scanner;

public class StaticBlockApp {
    static Scanner in = new Scanner(System.in);
    static int count;

    static {
        int num = in.nextInt();
        count = (num == 1) ? 1 : 0;
    }

    public StaticBlockApp() {
    }
```

- Στα static blocks ή static initializers μπορούμε να έχουμε περισσότερες εντολές και δομές ελέγχου ενώ στα static variables έχουμε μόνο απλά expressions
- Τόσο οι static μεταβλητές όσο και τα static blocks **εκτελούνται όταν φορτώνεται η κλάση στη μνήμη** και όχι όταν κάνουμε instantiate ένα instance της κλάσης

Εργασία (1)

- Δημιουργήστε ένα package model και μέσα σε αυτό ορίστε:
  - Μία κλάση User με πεδία id τύπου Long, firstname τύπου String, lastname τύπου String
  - Μία κλάση UserCredentials με πεδία id τύπου Long, username τύπου String και password τύπου String
- Ορίστε default constructors και overloaded constructors, getters και setters για όλα τα πεδία

Εργασία (2)

- Σε μία άλλη κλάση με όνομα Main δημιουργήστε ένα instance της κλάσης User και ένα instance της UserCredentials με τη χρήση overloaded constructors
- Εμφανίστε με *println* όλα τα πεδία των δύο instances σε μορφή {πεδίο1, πεδίο2, κλπ.}