ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ

Ενότητα 2: Βασικές εντολές

Μιχάλης Δρακόπουλος

Σχολή Θετικών επιστημών

Τμήμα Μαθηματικών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ (Java)
Ενότητα 2

Είσοδος από το πληκτρολόγιο:

1) Προσθήκη απαραίτητης βιβλιοθήκης στην κλάση μας:

```
import java.util.*;
```
στην αρχή της κλάσης

2) Δημιουργία αντικειμένου της κλάσης Scanner (με όνομα π.χ. input):

```
Scanner input = new Scanner(System.in);
```

3) Ερώτηση προς το χρήστη για να ζητήσουμε την είσοδο:

```
System.out.println("……");
```

4) Ανάγνωση δεδομένων από το πληκτρολόγιο:

```
int n = input.nextInt();          // για ακέραιους
```
ή
```
double x = input.nextDouble();    // για πραγματικούς
```
ή
```
String s = input.nextLine();      // για συμβολοσειρές
```

Παράδειγμα εισόδου / εξόδου

Το παρακάτω πρόγραμμα Java αποτελείται από μία μόνο κλάση, την κλάση εφαρμογής (περιέχει τη μέθοδο `main`). Δέχεται δύο ακέραιους αριθμούς από το χρήστη και εκτυπώνει στην οθόνη το άθροισμά τους.

```
/* A simple java input/output example.
   October 16, 2008 */

import java.util.*;      // "βιβλιοθήκη" που περιέχει την κλάση
                         // Scanner για είσοδο δεδομένων
public class Addition
{
   public static void main(String [] args)
   {
      System.out.println("Δώσε δύο ακέραιους αριθμούς:");
      int n1, n2;
      Scanner input = new Scanner(System.in);
      n1 = input.nextInt();
      n2 = input.nextInt();

      System.out.println("Το άθροισμα των δύο ακέραιων είναι:");
      System.out.println(n1+n2);

   }  // end of method

}  // end of class
```

Το κείμενο ανάμεσα στα σύμβολα `/*` και `*/` είναι *σχόλιο* και δεν αποτελεί μέρος του κώδικα της Java. Επίσης, το ίδιο συμβαίνει για το κείμενο που ακολουθεί το σύμβολο `//` σε μια γραμμή του προγράμματος (*σχόλιο γραμμής*).

- Σημείωση: Για την έξοδο στην οθόνη *δεν* χρειάζεται να προσθέσουμε κάποιο "import" στο πρόγραμμά μας. Ό,τι χρειάζεται για την έξοδο εισάγεται αυτόματα από τη Java. Αντίθετα, για την είσοδο, χρειάζεται να συμπεριλάβουμε (με το "import") τη «βιβλιοθήκη» (πακέτο) java.util στο πρόγραμμά μας. Περισσότερα για αυτή τη διαδικασία θα αναφερθούν σε επόμενη ενότητα.

Για να εκτελεσθεί το πρόγραμμά μας:

i) Το αποθηκεύουμε σε αρχείο: **Addition.java**
ii) Το κάνουμε compile: **javac Addition.java** → Addition.class (bytecode)
iii) Εκτελούμε το αρχείο bytecode με τον interpreter: java Addition

Κατά την εκτέλεσή του το πρόγραμμα θα ζητήσει την είσοδο δύο ακεραίων από το πληκτρολόγιο και κατόπιν θα εκτυπώσει στην οθόνη το άθροισμά τους.

Τελεστής εκχώρησης ή ανάθεσης (=)

```
<μεταβλητή> = <τιμή>;
```
ή
```
<μεταβλητή> = <έκφραση>;
```

Έκφραση: εντολή ή πράξη που το αποτέλεσμά της είναι κάποια τιμή.

- Υπολογίζεται η έκφραση δεξιά του = και η τιμή της αποθηκεύεται στη μεταβλητή που βρίσκεται αριστερά του =
- Η προηγούμενη τιμή της μεταβλητής χάνεται.

Ειδικοί τελεστές εκχώρησης (ή ανάθεσης)

Υπάρχουν και κάποιοι ειδικοί τελεστές εκχώρησης:

```
a += x; → a = a + x;
a -= x; → a = a - x;
a *= x; → a = a * x;
a /= x; → a = a / x;

a++; }
     } → a = a + 1;
++a; }

a--; }
     } → a = a - 1;
--a; }
```

Διαφορά μεταξύ a++ και ++a:

Το αποτέλεσμα των a++ και ++a είναι το ίδιο, με τη διαφορά ότι όταν βρίσκονται μέσα σε μία έκφραση, στην περίπτωση του a++ *πρώτα* γίνεται η πράξη (χρησιμοποιώντας την αρχική τιμή του a) και *μετά* αυξάνεται το a κατά 1, ενώ στην περίπτωση του ++a *πρώτα* γίνεται η αύξηση της τιμής του *a* κατά 1 και *μετά* γίνεται η πράξη.

π.χ.
```
int a, b, c;
a=2;
a+=6;     → a=8
a-=3;     → a=5
a*=2;     → a=10
b = 2 * a++;  → a=11, b=20
c = 2 * ++a;  → a=12, b=24
```

Ειδική περίπτωση:

- Τι συμβαίνει όταν στην ίδια έκφραση υπάρχει πάνω από μία φορά ο τελεστής ++ (ή --) στην ίδια μεταβλητή (ή αν η μεταβλητή με τον εν λόγω τελεστή επανεμφανίζεται απλά μέσα στην έκφραση):

π.χ., 1η περίπτωση:
```
int a = 5;
int x = (a++) + (++a);    // → x = 5 + 7 = 12,  a = 7
```

2η περίπτωση:
```
int a = 5;
int x = (++a) + (a++);    // → x = 6 + 6 = 12,  a = 7
```

Δηλαδή, στη περίπτωση του `++a` η μεταβλητή πρώτα αυξάνεται κατά 1 και μετά παίρνει μέρος στην έκφραση, ενώ στην περίπτωση του `a++` παίρνει μέρος στην έκφραση με την αρχική της τιμή και αμέσως μετά αυξάνεται κατά 1, *οπότε αν υπάρχει και σε άλλο σημείο της έκφρασης, παίρνει μέρος με τη νέα της πλέον τιμή*.

Συμβατότητες εκχώρησης και αυτόματη μετατροπή τύπου

```
double doubleVar;
doubleVar = 7;   // ok: το 7 που είναι int μετατρέπεται σε double
```

Θα ήταν το ίδιο αν γράφαμε:
```
doubleVar = 7.0;
```

Επίσης:

```
int intVar;
intVar = 7;
double doubleVar;
doubleVar = intVar;  // ok: το ίδιο με πριν αλλά με μεταβλητή int
```

Γενικά, μια τιμή οποιουδήποτε τύπου μπορεί να εκχωρηθεί σε μια μεταβλητή οποιουδήποτε τύπου που βρίσκεται δεξιά από αυτόν στη λίστα:

`byte` → `short` → `int` → `long` → `float` → `double`

δηλαδή, μια τιμή τύπου `byte` μπορεί να μπει σε μια μεταβλητή τύπου `int`, ενώ μια τιμή τύπου `float` δεν μπορεί να μπει σε μια μεταβλητή τύπου `short`. κτλ.

(Μια τιμή τύπου `char` εκχωρείται σε μεταβλητή από `int` και πάνω)

Αριθμητικοί τελεστές και αυτόματη μετατροπή τύπου

Πράξεις: +, -, *, /, % (υπόλοιπο διαίρεσης)

- Το αποτέλεσμα μιας πράξης μεταξύ 2 μεταβλητών του ίδιου τύπου, είναι προφανώς του ίδιου τύπου με αυτόν των μεταβλητών
- Όταν 2 μεταβλητές είναι διαφορετικού τύπου, το αποτέλεσμα είναι του πιο «σύνθετου» τύπου (δηλ., του τύπου με τη μεγαλύτερη ακρίβεια):

`byte` → `short` → `int` → `long` → `float` → `double`

- Οι μεγαλύτερες εκφράσεις (πράξεις μεταξύ περισσότερων από 2 μεταβλητών) μπορούν πάντα να αναλυθούν σε βήματα με πράξεις 2 μεταβλητών, άρα το αποτέλεσμά τους είναι του πιο «σύνθετου» τύπου μεταξύ των τύπων όλων των μεταβλητών της έκφρασης.

π.χ., 3 * 4 → 12, 3 % 4 → 3, 11 % 3 → 2, 3 / 4 → 0 (γιατί είναι ακέραιοι!),

Μετατροπή τύπου (type casting)

Στις προηγούμενες περιπτώσεις η μετατροπή τύπου γινόταν αυτόματα. Όμως:

```
double distance;
distance = 9.0;
int points;
points = distance;   → ΣΦΑΛΜΑ!  (run-time error)
```

Το σωστό θα ήταν:

```
points = (int)distance;
```

Το `(int)` μετατρέπει τη μεταβλητή `distance` σε `int`. Στην ουσία δεν αλλάζει ο τύπος της μεταβλητής `distance`, η οποία παραμένει `double`. Αλλάζει απλά ο τύπος της τιμής της, ώστε να μπορεί στη συνέχεια να αποθηκευτεί σε μία αντίστοιχου τύπου μεταβλητή, όπως εδώ η `points`. Άρα, η `points` είναι η «ακέραια έκδοση» της τιμής της `distance`.

ΠΡΟΣΟΧΗ: Η `double` τιμή της μεταβλητής `distance` δεν αλλάζει. Η `(int)` δημιουργεί μια ακέραια τιμή βάσει της `double` τιμής. Άρα δεν στρογγυλοποιείται η `double` τιμή στην αντίστοιχη `int`, αλλά απλά χάνεται το δεκαδικό της μέρος. Έτσι,

- αν η `distance` = 12.29 η `(int)distance` → 12
- αν η `distance` = 12.99 η `(int)distance` → 12 πάλι (απλά χάνεται το δεκαδικό μέρος και μένει το ακέραιο μέρος).

Γενικά, η σύνταξη μετατροπής τύπου είναι:

```
(τύπος)μεταβλητή;
ή
(τύπος)έκφραση;
```

Άρα, στο παράδειγμα του προηγούμενου μέρους που η διαίρεση των ακεραίων 3/4 έδωσε αποτέλεσμα 0 επειδή και το αποτέλεσμα είναι ακέραιος, εάν δε θέλαμε να χάσουμε την ακρίβεια της πράξης, θα έπρεπε να μετατρέψουμε τουλάχιστον τον έναν από τους δύο int σε double:

`(double)3 / 4` → `0.75` ή `3 / (double)4` → `0.75`

ή απλά να γράψουμε τον ένα αριθμό σε μορφή πραγματικού:

3.0 / 4 → 0.75 ή 3 / 4.0 → 0.75

Μετατροπή τύπου `char` σε `int` και το αντίστροφο:

```
char symbol;
symbol = '7';
System.out.println((int)symbol);  →  55
```

Τυπώνει 55 γιατί ο *χαρακτήρας* 7 της μεταβλητής `symbol` είναι ένας χαρακτήρας Unicode και το 55 είναι η θέση του στη λίστα των χαρακτήρων αυτών. Δηλαδή η μετατροπή σε `int` γίνεται μέσω της αντιστοιχίας του Unicode. Το `'7'` εδώ δεν είναι ένα νούμερο, είναι απλά ένας χαρακτήρας, ο no. 55 του συνόλου χαρακτήρων Unicode. Κάθε χαρακτήρας λοιπόν αντιστοιχεί σε έναν ακέραιο, τη θέση του στον πίνακα των χαρακτήρων Unicode. Χρειάζεται ιδιαίτερη προσοχή στη διαχείριση χαρακτήρων και στις πράξεις μεταξύ χαρακτήρων σε μια εντολή `System.out.println`.

Η εντολή `System.out.println('a');` θα τυπώσει: `a`
όμως η εντολή `System.out.println('a' + 'b');` θα τυπώσει `195`. Αυτό συμβαίνει γιατί με τον τελεστή + στην ουσία οι χαρακτήρες μετατρέπονται στους αντίστοιχους ακέραιους (97 και 98 αντίστοιχα) και έτσι πραγματοποιείται η πρόσθεση.

Η εντολή `System.out.println('a' + 1);` εκτυπώνει `98`. Εάν κάποιος θέλει να εκτυπώσει τον επόμενο χαρακτήρα του `'a'` και όχι το άθροισμα 97+1, τότε θα πρέπει να *μετατρέψει* το άθροισμα σε `char`: `System.out.println(`**`(char)`**`('a' + 1));`. Η εντολή αυτή εκτυπώνει: `b`.

Κανόνες προτεραιότητας πράξεων

Τρόπος για τον έλεγχο της προτεραιότητας: παρενθέσεις.

Οι παρενθέσεις έχουν προτεραιότητα στις πράξεις, ξεκινώντας από τις πιο εσωτερικές.

Όταν δεν ελέγχεται η προτεραιότητα από τον προγραμματιστή, η Java αποφασίζει βάσει συγκεκριμένων κανόνων, δηλαδή με βάση την ακόλουθη σειρά προτεραιότητας:

i) μοναδιαίοι τελεστές: - *(το πρόσημο)*, ++, --, ! *(η άρνηση - περισσότερα σε επόμενο μάθημα)*
ii) δυαδικοί τελεστές *, /, %
iii) δυαδικοί τελεστές +, -

(*μοναδιαίοι* τελεστές ονομάζονται αυτοί που εφαρμόζονται σε μία μόνο μεταβλητή, ενώ *δυαδικοί* αυτοί που εφαρμόζονται σε δύο μεταβλητές)

π.χ., το $\frac{a-b}{c+d}$ γράφεται: `(a - b) / (c + d)`

Αν το γράφαμε χωρίς παρενθέσεις, δηλαδή: `a - b / c + d` , τότε θα αντιπροσώπευε το $a-\frac{b}{c}+d$ (η διαίρεση b/c θα γινόταν πρώτη).

$\frac{x}{y+5z}$ → `x / (y + 5 * z)` ή `x / (y + (5 * z))`

επειδή όμως ο πολλαπλασιασμός προηγείται της πρόσθεσης, η extra παρένθεση στο (5*z) δεν είναι απαραίτητη (χωρίς φυσικά να είναι λάθος εάν υπάρχει).

Μαθηματικές συναρτήσεις

Math.sin(x) → ημ(x), Math.cos(x) → συν(x), Math.exp(x) → e^x, Math.pow(x,y) → x^y

Math.sqrt(x) → √x, Math.abs(x) → |x|, Math.log(x) → $\log_e x$, Math.log10(x) → $\log_{10} x$, Math.PI → π

`Math.sin(x)` → ημ(x), `Math.cos(x)` → συν(x), `Math.exp(x)` → e^x,
`Math.pow(x,y)` → x^y, `Math.sqrt(x)` → √x, `Math.abs(x)` → |x|,
`Math.log(x)` → $\log_e x$, `Math.log10(x)` → $\log_{10} x$, `Math.PI` → π

Άλλο ένα παράδειγμα προγράμματος Java

Μας ζητείται να γράψουμε πρόγραμμα Java που να μετατρέπει ένα ποσό x λεπτών (cents) του ευρώ (όπου το x είναι ακέραιο ποσό μεταξύ 1 και 99) σε όσο το δυνατόν λιγότερα κέρματα των 50c, 20c, 10c, 5c, 2c και 1c (c = cents = λεπτά).

→ Τι μεταβλητές χρειαζόμαστε;

- έχουμε ένα αρχικό ποσό (1-99 cents), άρα θέλουμε μια int μεταβλητή: `originalAmount`
- θα μετράμε το πλήθος του κάθε κέρματος, άρα θέλουμε μια int μεταβλητή για κάθε κέρμα: `coin50`, `coin20`, `coin10`, `coin5`, `coin2` και `coin1`.

Αυτές προς το παρόν.

→ Πως θα υπολογίσουμε τα κέρματα;

- Πόσα 50c υπάρχουν στο αρχικό ποσό;
- Τι ποσό απομένει αφού αφαιρέσουμε τα κέρματα των 50c;
  - Άρα χρειαζόμαστε κι άλλη μεταβλητή! Για να κρατάμε το ποσό που απομένει κάθε φορά. → μεταβλητή `amount`
- κτλ, για 20c, 10c, 5c, 2c, 1c.

Ο κώδικας Java:

```
import java.util.*;  // "βιβλιοθήκη" για user input
public class ChangeMaker
{
   public static void main(String [] args)
   {
      int originalAmount, amount, coin50, coin20, coin10, coin5, coin2, coin1;
      System.out.println("Δώσε το ποσό των λεπτών (από 1 ως 99)");
      Scanner input = new Scanner(System.in);
      originalAmount = input.nextInt();

      amount = originalAmount;
      coin50 = amount / 50;  // δουλεύει γιατί η amount είναι int
      amount = amount % 50;  // το υπόλοιπο ποσό
      coin20 = amount / 20;
      amount = amount % 20;
      coin20 = amount / 10;
      amount = amount % 10;
      coin20 = amount / 5;
      amount = amount % 5;
      coin20 = amount / 2;
      amount = amount % 2;
      coin1 = amount;        // το τελικό ποσό που απομένει

      System.out.println(originalAmount + " λεπτά σε κέρματα είναι:");
      System.out.println(coin50 + " 50λεπτα " + coin20 + " 20λεπτα " + coin10 +
       " 10λεπτα " + coin5 + " 5λεπτα " + coin2 + " 2λεπτα και " coin1 +
       " λεπτά.");

   }   // end of main

}   // end of class
```

Η Κλάση `String`

Κάθε συμβολοσειρά (αλφαριθμητικό) είναι αντικείμενο της κλάσης `String` και όχι μια μεταβλητή πρωτογενούς τύπου (όπως π.χ. η `double`, η `int`, κτλ.).

Δημιουργία: `String str = "Hello";`

Τελεστής συνένωσης (δημιουργία νέων String):

```
String s1 = "Hello ";
String s2 = "there";
String s3 = s1 + s2;
System.out.println(s3);   // → Hello there
```

- *«Πρόσθεση» (συνένωση) String με άλλους τύπους δεδομένων:*

```
System.out.println(4+7);  // → Τυπώνει: 11
System.out.println("Το άθροισμα είναι: " + 4 + 7);
     // → Τυπώνει: Το άθροισμα είναι: 47
```

Για να τυπωθεί το επιθυμητό 11, θα πρέπει να χρησιμοποιήσουμε παρένθεση στην πρόσθεση 4+7, ως εξής:

```
System.out.println("Το άθροισμα είναι: " + (4 + 7));
     // → Τυπώνει: Το άθροισμα είναι: 11
```

(Περισσότερα για την κλάση `String` θα αναφερθούν σε επόμενη ενότητα)

Σύγκριση μεταβλητών

Η σύγκριση δύο μεταβλητών στη Java γίνεται με τους ακόλουθους τελεστές (δεξιά στήλη):

Μαθηματικά	*Java*
=	==
<	<
≤	<=
>	>
≥	>=
≠	!=

- Το αποτέλεσμα μιας σύγκρισης είναι τύπου `boolean` (`true` ή `false`)

π.χ.,

```
int a=5, b=5;
boolean c, d;
c = (a == b);  //  → c=true
d = (a > b);   //  → d=false
```

Λογικές σχέσεις

- Σύζευξη (AND): `&&`
- Διάζευξη (OR): `||`
- Άρνηση (NOT): `!`

- Λογικές σχέσεις γίνονται μεταξύ `boolean` μεταβλητών ή εκφράσεων που έχουν `boolean` αποτέλεσμα (τιμή).
- Το αποτέλεσμά τους είναι και αυτό `boolean` (`true` ή `false`).

Πίνακας αληθείας:

p	q	p&&q	p||q
True	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

- Ο υπολογισμός των λογικών σχέσεων σταματάει τη στιγμή που προσδιορίζεται η τιμή τους (υπάρχει τρόπος να *μη* γίνεται αυτό, με τα σύμβολα `&` και `|`). Δηλαδή, στη λογική σχέση: A&&B, αν το A είναι `false`, τότε δεν ελέγχεται καν το B, και το αποτέλεσμα του A&&B γίνεται `false`. Αντίστοιχα, στο A || B, αν το A είναι `true`, το αποτέλεσμα γίνεται `true` χωρίς να ελεγχθεί το B.

Άρα, η ακόλουθη έκφραση δε δημιουργεί πρόβλημα:

```
(3 == 7) && (2 == (3 / 0) );          → βγάζει false
```

(γιατί 3 == 7 είναι `false` και δε μπαίνει ποτέ στο 3/0 του δεύτερου μέρους που κανονικά είναι σφάλμα (διαίρεση με το μηδέν)).

Αντίθετα, αν ήταν:

```
(7 == 7) && (2 == (3 / 0) );
```

θα έβγαζε *σφάλμα* (run-time error: division by zero).

Άρα, είναι επιθυμητό κατά τη δημιουργία λογικών σχέσεων, η απλούστερη έκφραση να γράφεται πρώτη, όπου αυτό είναι δυνατό.

Οι αντίστοιχοι τελεστές `&` και `|` ελέγχουν και τις δύο εκφράσεις, ανεξάρτητα από το αποτέλεσμα της πρώτης έκφρασης. Άρα, η έκφραση που παραπάνω έβγαζε `false`, με χρήση του `&` θα είναι εσφαλμένη, αφού θα ελέγξει και τη δεύτερη, λανθασμένη έκφραση:

```
(3 == 7) & (2 == (3 / 0) );      → run-time error: division by zero
```

Συχνά λάθη:

- Το x δεν ισούται με 2 ή 3:

Λάθος: `x!=2 || x!=3` // → Είναι πάντα `true`!

Σωστό: `x!=2 && x!=3` // ή: `!(x==2 || x==3)`

Ισχύουν οι ιδιότητες: → Το: `!(p || q)` ταυτίζεται με το: `!p && !q`
→ Το: `!(p && q)` ταυτίζεται με το: `!p || !q`

- Το: $0 \leq x < 1$ γράφεται: `0 <= x && x < 1`

(δεν συγκρίνουμε πάνω από δύο μεταβλητές μαζί)

Παράδειγμα:

- Για δεδομένο έτος (year) να δοθεί τιμή στην boolean μεταβλητή isLeapYear ανάλογα με το αν το έτος είναι δίσεκτο (true) ή όχι (false):

Λύση:

Πότε ένα έτος είναι δίσεκτο;
Δίσεκτα είναι τα έτη που διαιρούνται με το 4, *εκτός* από τα έτη των αιώνων (π.χ., 1700, 1800, 1900, κτλ.). Κατ' εξαίρεση, *είναι* δίσεκτα τα έτη των αιώνων που διαιρούνται με το 400 (π.χ., 1600, 2000, 2400, κτλ.).

Άρα, ένα έτος είναι δίσεκτο:

α) αν διαιρείται με το 4 αλλά όχι με το 100
ή
β) αν διαιρείται με το 400

Οπότε:

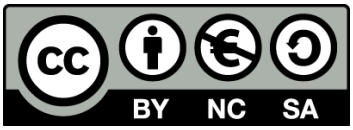
```
int year;
boolean isLeapYear;
.
.
isLeapYear = ((year%4==0) && (year%100!=0)) || (year%400==0);

// ή καλύτερα, πρώτα η απλούστερη έκφραση (year%400==0)
// και μετά η σύζευξη:
// isLeapYear = (year%400==0) || ((year%4==0) && (year%100!=0));
```

Σημειώματα

Σημείωμα Αναφοράς

Μιχάλης Δρακόπουλος. «Πληροφορική ΙΙ. Ενότητα 2: Βασικές εντολές». Έκδοση: 1.0. Αθήνα 2014. Διαθέσιμο από τη δικτυακή διεύθυνση:
http://opencourses.uoa.gr/courses/MATH106/.

Σημείωμα Αδειοδότησης



[1] http://creativecommons.org/licenses/by-nc-sa/4.0/

Ως **Μη Εμπορική** ορίζεται η χρήση:

- που δεν περιλαμβάνει άμεσο ή έμμεσο οικονομικό όφελος από την χρήση του έργου, για το διανομέα του έργου και αδειοδόχο
- που δεν περιλαμβάνει οικονομική συναλλαγή ως προϋπόθεση για τη χρήση ή πρόσβαση στο έργο
- που δεν προσπορίζει στο διανομέα του έργου και αδειοδόχο έμμεσο οικονομικό όφελος (π.χ. διαφημίσεις) από την προβολή του έργου σε διαδικτυακό τόπο

Ο δικαιούχος μπορεί να παρέχει στον αδειοδόχο ξεχωριστή άδεια να χρησιμοποιεί το έργο για εμπορική χρήση, εφόσον αυτό του ζητηθεί.

Διατήρηση Σημειωμάτων

- Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:
- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στο πλαίσιο του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αθηνών**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.