ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ

Ενότητα 6: Pass-by-value και φαινομενικό pass-by-reference

Μιχάλης Δρακόπουλος

Σχολή Θετικών επιστημών

Τμήμα Μαθηματικών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ (Java)
Ενότητα 6

Pass-by-value και φαινομενικό pass-by-reference

Γενικά, στις γλώσσες προγραμματισμού, οι όροι "pass-by-value" (πέρασμα με τιμή) και "pass-by-reference" (πέρασμα με αναφορά) αναφέρονται στον τρόπο με τον οποίο περνάει μία μέθοδος (που καλεί κάποια άλλη) τα ορίσματα εισόδου στην καλούμενη μέθοδο. Ένα τέτοιο πέρασμα (pass) μπορεί να γίνει είτε περνώντας τις τιμές των ορισμάτων αυτών στην καλούμενη μέθοδο (pass-by-value), είτε περνώντας αναφορές (στη θέση μνήμης) των ορισμάτων αυτών (pass-by-reference).

Έστω μια κλάση υποστήριξης `ClassB`:

```
public class ClassB
{
   private int x, y;
   public ClassB(int x, int y)
   {
      this.x = x;
      this.y = y;
   }

   public void setValues(int a, int b)
   {
      x = a;  y = b;
   }

   public int getX()
   {
      return x;
   }
   public int getY()
   {
      return y;
   }
}
```

και η κλάση εφαρμογής `Program` που περιέχει τη `main` και μία μέθοδο `add5`:

```
public class Program
{
   public static void main (String[] args)
   {
      ClassB obj = new ClassB(5,10);
      int z = 100;
      System.out.println("Τιμές των x, y και z πριν την κλήση της μεθόδου:");
      System.out.println("x = " + obj.getX());
      System.out.println("y = " + obj.getY());
      System.out.println("z = " + z);

      add5(obj,z);   // Κλήση της add5 που ορίζεται παρακάτω
      System.out.println("Τιμές των x, y και z μετά την κλήση της μεθόδου:");
      System.out.println("x = " + obj.getX());
      System.out.println("y = " + obj.getY());
      System.out.println("z = " + z);
   }
```

```
   public static void add5(ClassB obj, int z)
   {
      obj.setValues(obj.getX()+5, obj.getY()+5);
      z+=5;
   }
}
```

Το πρόγραμμα αυτό θα εκτυπώσει τα εξής:

```
Τιμές των x, y και z πριν την κλήση της μεθόδου:
x = 5
y = 10
z = 100
Τιμές των x, y και z μετά την κλήση της μεθόδου:
x = 10
y = 15
z = 100
```

Δηλαδή, η τιμή της μεταβλητής `z` δεν άλλαξε μετά την κλήση της μεθόδου `add5`, ενώ αντιθέτως οι τιμές των μεταβλητών `x` και `y` του αντικειμένου `obj` της κλάσης `ClassB` άλλαξαν.

Στη Java όλες οι παράμετροι περνιούνται με τιμή (pass-by-value) από τη μέθοδο που κάνει την κλήση στη μέθοδο που καλείται. Αυτό ισχύει *και* για τις μεταβλητές *και* για τα αντικείμενα. Δηλαδή, και τα αντικείμενα περνιούνται με τιμή (pass-by-value). Θεωρητικά στη Java δεν υπάρχει πέρασμα με αναφορά (pass-by-reference) όπως υπάρχει σε άλλες γλώσσες προγραμματισμού.

Τότε γιατί στο παραπάνω παράδειγμα φαίνεται να γίνεται pass-by-reference στην περίπτωση του αντικειμένου; Αυτό συμβαίνει γιατί στη Java το κάθε αντικείμενο ουσιαστικά είναι μια αναφορά στη θέση μνήμης που περιέχονται τα στοιχεία (δεδομένα) του αντικειμένου.

Για να κατανοήσει κανείς τι ακριβώς συμβαίνει με τα αντικείμενα, πρέπει καταρχάς να κατανοήσει τι συμβαίνει όταν μια απλή μεταβλητή περνιέται με τιμή κατά την κλήση μιας μεθόδου. Όταν καλείται μια μέθοδος που δέχεται μια παράμετρο εισόδου, η μεταβλητή που δίνεται από τη μέθοδο που κάνει την κλήση στη θέση της παραμέτρου, αντιγράφεται σε μία νέα προσωρινή μεταβλητή που θα χρησιμοποιηθεί ως παράμετρος για την κλήση.

Το ίδιο συμβαίνει και στην περίπτωση που αντί για μεταβλητή περνιέται ένα αντικείμενο σαν παράμετρος εισόδου στην καλούμενη μέθοδο: το αντικείμενο αντιγράφεται σε ένα προσωρινό αντικείμενο που αποστέλλεται στη μέθοδο που καλείται. *Όμως, αυτό το προσωρινό αντικείμενο δεν είναι τίποτε άλλο από μια διαφορετική αναφορά στο ίδιο αρχικό αντικείμενο* (αφού τα αντικείμενα είναι αναφορές στις θέσεις μνήμης που περιέχονται τα αντικείμενα). Επομένως, ό,τι αλλαγές γίνουν στα δεδομένα του αντικειμένου από την κληθείσα μέθοδο, θα ισχύουν και για το αντικείμενο που περάστηκε ως παράμετρος στην καλούσα μέθοδο, αφού πρόκειται για το ίδιο αντικείμενο.
Για αυτό το λόγο, στο παραπάνω παράδειγμα οι μεταβλητές του αντικειμένου `obj` παραμένουν αλλαγμένες μετά την επαναφορά της ροής του κώδικα από τη μέθοδο `add5` στη μέθοδο `main` (μετά την κλήση της μεθόδου `add5`), σε αντίθεση με την απλή μεταβλητή `z`, η οποία παραμένει αμετάβλητη στην αρχική της τιμή.

Σημείωση: Επειδή στη Java οι **πίνακες** (που θα αναφερθούν σε επόμενο κεφάλαιο) είναι στην ουσία αντικείμενα, ό,τι ισχύει για τα αντικείμενα όσον αφορά το πέρασμα των τιμών τους, ισχύει και για αυτούς.

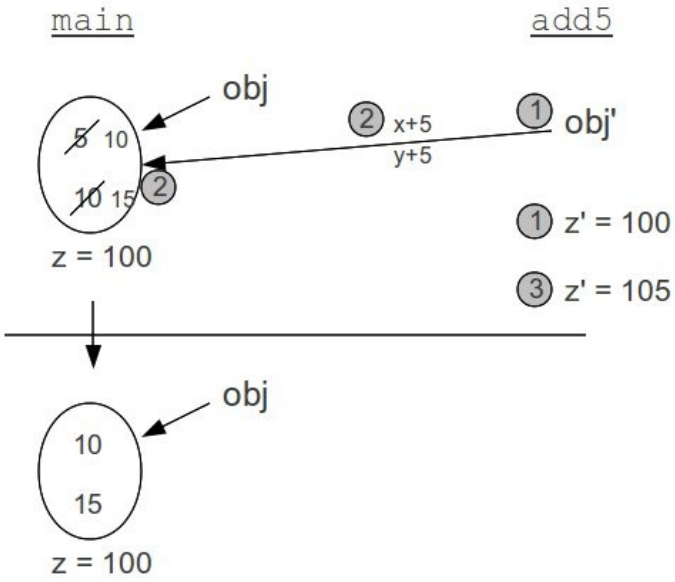
ΠΡΟΣΟΧΗ: Αν μέσα στη μέθοδο `add5` ή σε κάποια άλλη αντίστοιχη μέθοδο δεν άλλαζαν οι τιμές μεταβλητών του αντικειμένου `obj`, αλλά άλλαζε *ολόκληρο* το αντικείμενο (σε αντιστοιχία με την αλλαγή μιας απλής μεταβλητής), τότε η αλλαγή αυτή θα έμενε στη μέθοδο και δε θα "πέρναγε" στη μέθοδο `main`, δηλαδή θα συνέβαινε ό,τι ακριβώς συμβαίνει και με τις απλές μεταβλητές. Αυτό είναι μια "απόδειξη" ότι ***και*** *τα αντικείμενα ουσιαστικά γίνονται pass-by-value στη Java*. Π.χ., αν στο παραπάνω παράδειγμα, στην κλάση `Program` υπήρχε αντί της μεθόδου `add5` η μέθοδος `change`, οπότε η `Program` γινόταν ως εξής:

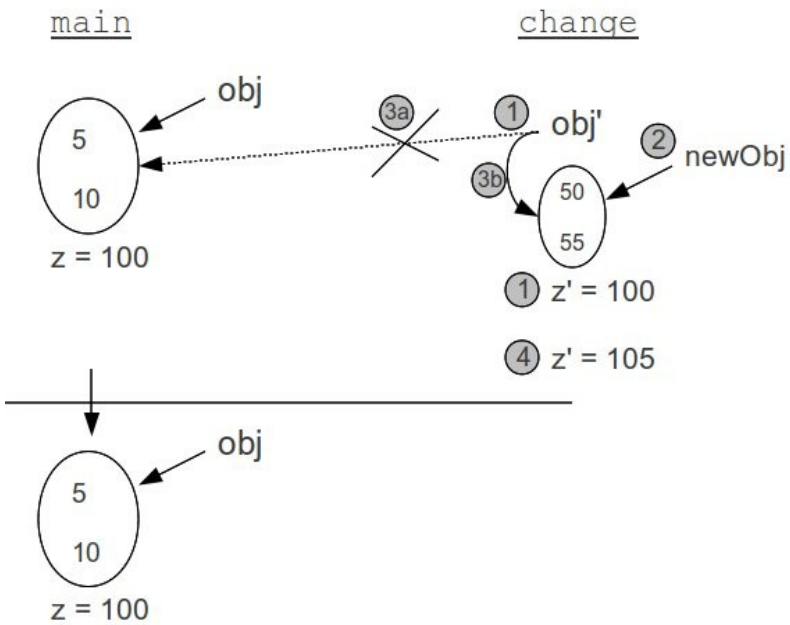
```
public class Program
{
   public static void main (String[] args)
   {
      ClassB obj = new ClassB(5,10);
      int z = 100;
      System.out.println("Τιμές των x, y και z πριν την κλήση της μεθόδου:");
      System.out.println("x = " + obj.getX());
      System.out.println("y = " + obj.getY());
      System.out.println("z = " + z);

      change(obj,z);    // Κλήση της change που ορίζεται παρακάτω

      System.out.println("Τιμές των x, y και z μετά την κλήση της μεθόδου:");
      System.out.println("x = " + obj.getX());
      System.out.println("y = " + obj.getY());
      System.out.println("z = " + z);
   }

   public static void change(ClassB obj, int z)
   {
      ClassB newObj = new ClassB(50,55);
      obj = newObj;
      z+=5;
   }
}
```

τότε το πρόγραμμα θα εκτύπωνε τα εξής:

```
Τιμές των x, y και z πριν την κλήση της μεθόδου:
x = 5
y = 10
z = 100
Τιμές των x, y και z μετά την κλήση της μεθόδου:
x = 5
y = 10
z = 100
```

Πλέον η συμπεριφορά των τιμών των `x` και `y` του αντικειμένου `obj` είναι η ίδια με αυτή μιας κανονικής μεταβλητής, δηλαδή παραμένουν στις προηγούμενες τιμές τους (5 και 10 αντίστοιχα), πράγμα που σημαίνει ότι όντως το αντικείμενο `obj` περάστηκε με τιμή (pass-by-value) στη μέθοδο `change` και δε μεταβλήθηκε από αυτή. Το μόνο που άλλαξε σε σχέση με πριν είναι ότι πλέον η μέθοδος δεν τροποποιεί μεμονωμένα τις τιμές των μεταβλητών του αντικειμένου που δέχεται, αλλά τροποποιεί "ολόκληρο" το αντικείμενο, άρα το μόνο που τροποποιεί είναι την αναφορά στη θέση μνήμης του προσωρινού αντικειμένου που δημιουργείται, το οποίο τώρα απλά αναφέρεται στη θέση μνήμης ενός νέου αντικειμένου (με τιμές μεταβλητών 50 και 55). Οι μεταβλητές του αρχικού αντικειμένου `obj` φυσικά παραμένουν αμετάβλητες μέσα στη μέθοδο `main`, όπως ακριβώς συμβαίνει και με τη μεταβλητή `z`. Το τι ακριβώς γίνεται στα δύο αυτά παραδείγματα, φαίνεται σχηματικά παρακάτω.





Η Κλάση `String`

Κάθε συμβολοσειρά (ή αλφαριθμητικό) (string) είναι αντικείμενο της κλάσης `String`.

Βασική ιδιότητα: Τα string, δηλ. τα αντικείμενα της κλάσης `String`, δεν τροποποιούνται.

Τρόποι δημιουργίας συμβολοσειρών:

- Με τη χρήση εισαγωγικών: `String str = "Hello "`
- Με τη χρήση των + ή += πάνω σε υπάρχοντα strings
  `"Hello " + "world"` → νέο string → `Hello world`
- Με κανονική δημιουργία αντικειμένου μέσω της `new`.

Η `String` έχει δύο κατασκευαστές (constructors):

```
public String()  και  public String(String value)
```

π.χ.
```
String str1 = new String();
String str2 = new String("Hello");
```

Προσοχή στον τελεστή + :

```
double x=15, y=25;
System.out.println(x+y);
System.out.println("Το άθροισμα είναι " + x + y);
System.out.println("Το άθροισμα είναι " + (x + y));
```

Ο κώδικας αυτός θα εκτυπώσει:

```
40.0
Το άθροισμα είναι 15.025.0
Το άθροισμα είναι 40.0
```

Δηλαδή, όταν ο τελεστής + βρει κάποιο string, προσθέτει απλά τις τιμές των μεταβλητών που βρίσκει μετά σαν συνέχεια στο string αυτό. Άρα, για να γίνει η πράξη της πρόσθεσης των δύο `double` μεταβλητών, θα πρέπει να μπει σε παρένθεση για να προηγηθεί της συνένωσης με το string.

Βασικές μέθοδοι της κλάσης `String`

- `public int length()` → επιστρέφει το πλήθος των χαρακτήρων του αλφ/κού

π.χ.
```
String a = "blah blah";
int b = a.length();   // → b = 9
```

- `public char charAt(int index)` → επιστρέφει τον χαρακτήρα στη θέση `index`

```
char d = a.charAt(3);  // → d = 'h'   (μετράμε από το 0!)
```

- `public int indexOf(char ch)` → επιστρέφει την πρώτη θέση του `ch`

```
a.indexOf('a');  → 2
```

- `public int indexOf(char ch, int start)` → επιστρέφει την πρώτη θέση του `ch` από τη θέση `start` και μετά.

```
a.indexOf('a',3);  → 7
```

- `public int indexOf(String str)` → επιστρέφει την πρώτη θέση του `str`

- `public int indexOf(String str, int start)` → επιστρέφει την πρώτη θέση του `str` από τη θέση `start` και μετά.

- `public int lastIndexOf(char ch)` → επιστρέφει την τελευταία θέση του `ch`

- `public int lastIndexOf(char ch, int start)` → επιστρέφει την τελευταία μέχρι και το `start` θέση του `ch`

- `...` [αντίστοιχα και για παράμετρο `String str`].

Όλες αυτές οι μέθοδοι αν δεν βρουν αυτό που ψάχνουν στο string (το `ch` ή το `str`), επιστρέφουν **`-1`**.

Μέθοδοι σύγκρισης

- `public boolean equals(String str)`
  - → `true`, αν βρει ίδιο μήκος και ακριβώς ίδιους χαρακτήρες
  - → `false`, σε διαφορετική περίπτωση

π.χ.
```
String e = "blah blah";
boolean f = a.equals(e);  // → f = true
```

Ο τελεστής == όταν εφαρμοστεί σε δύο string συγκρίνει τις θέσεις μνήμης τους και όχι το περιεχόμενό τους.

- `public boolean equalsIgnoreCase(String str)`
  Αγνοεί κεφαλαία ή πεζά γράμματα

- `public int compareTo(String str)`
  - → `< 0` αν κάποιο string έχει < μήκος του `str`
  - → `= 0` αν κάποιο string έχει = μήκος με το `str`
  - → `> 0` αν κάποιο string έχει > μήκος του `str`

```
String h = "blah";
int i = h.compareTo(a);  // → i<0
```

```
- public boolean regionMatches(int start, String str,
                               int strStart, int len)

          h.regionMatches(1,a,6,3);  // → true

               h →  b l a h
                    0 1 2 3

               a →   b l a h   b l a h
                     0 1 2 3 4 5 6 7 8

boolean b = a.regionMatches(2,"Ah",0,2);  // → false  (ah ≠ Ah)

- public boolean regionMatches(boolean ignoreCase, int start,
                               String str, int strStart, int len)

boolean b = a.regionMatches(true,2,"Ah",0,2);  // → true
```

Μέθοδοι ελέγχου αρχής/τέλους:

```
- public boolean startsWith(String prefix)

- public boolean endsWith(String suffix)

     boolean b = a.endsWith("ah")  → true
```

Μέθοδοι δημιουργίας νέων αντικειμένων `String`

```
- public String replace(char oldChar, char newChar)

          String j = a.replace('a','i');  // → blih blih
```

Δεν αλλάζει το string `a`. Φτιάχνει **νέο** string (αντικείμενο)

```
- public String toLowerCase()
```

Παράδειγμα: Στην περίπτωση απλών αντικειμένων π.χ. μιας κλάσης υποστήριξης `SuppClass`, θα μπορούσαμε να έχουμε τον εξής κώδικα σε μια άλλη κλάση:

```
// Δημιουργία αντικειμένου της SuppClass
SuppClass obj1 = new SuppClass();

// Απόδοση τιμής στη μεταβλητή x του obj1
obj1.x = 10;

// Δημιουργία αντίγραφου του αντικειμένου obj1
SuppClass obj2 = obj1;

// Απόδοση τιμής στη μεταβλητή x του obj2
obj2.x = 5;
```

```
// Εκτύπωση στην οθόνη του x του obj1
System.out.println(obj1.x);
```

Ο κώδικας αυτός θα εκτυπώσει την τιμή **5**.

Στην ουσία έχουμε **ένα** αντικείμενο της `SuppClass`, το οποίο έχει δύο ονόματα, `obj1` και `obj2`. Οπότε, αλλάζοντας την τιμή του `x` στο `obj2`, στην ουσία αλλάζει η τιμή και στο `obj1`.

Αν τώρα έχουμε έναν αντίστοιχο κώδικα, αλλά με αντικείμενα της κλάσης `String` αντί κάποιας άλλης κλάσης:

```
// Δημιουργία αντικειμένου «τύπου» String
String k = "Hello";

// Δημιουργία αντίγραφου του αντικειμένου k
String m = k;

// «Τροποποίηση» του αντικειμένου m
m = m.toLowerCase();

// Εκτύπωση στην οθόνη της τιμής του αντικειμένου k
System.out.println(k);
```

Ο κώδικας αυτός θα εκτυπώσει **`Hello`** και όχι `hello` όπως θα περίμενε κάποιος, σε αντιστοιχία με το προηγούμενο παράδειγμα με την `SuppClass`.

Αυτό συμβαίνει γιατί κατά την εκτέλεση της εντολής `m.toLowerCase()` δεν αλλάζει η τιμή του `m` και άρα και του `k`, αλλά δημιουργείται ένα **νέο** αντικείμενο, στο οποίο «δείχνει» πλέον το αντικείμενο `m`. **Άρα, το αρχικό αντικείμενο `k` μένει αμετάβλητο (`Hello`), το `m` παύει να υπάρχει ως αντίγραφο του `k` και δημιουργείται ένα νέο αντικείμενο (`hello`) με το όνομα `m`.**

Τα αντικείμενα τύπου `String` λοιπόν, δεν μπορεί να μεταβληθούν. Κάθε «τροποποίηση» επιστρέφει ένα <u>νέο αντικείμενο</u> τύπου `String`.

- `public String toUpperCase()`

- `public String trim()` → κόβει τα κενά σε αρχή και τέλος

- `public String concat(String str)` → το ίδιο με το `+`

```
          // Apo prin: a=blah blah kai h=blah
    String s = a.concat(h); // → blah blahblah   (ίδιο με το a+h;)
```

Μετατροπές από/σε `String`

Από μεταβλητή τύπου `boolean`, `int`, `long`, `float`, `double`, `char` σε `String`:

```
String.valueOf(<μεταβλητή>);
```

π.χ.,

```
int n = 10;
String p = String.valueOf(n);  // → p = "10"
```

Από `String` σε `boolean` → `Boolean.parseBoolean(str)`
σε `int` → `Integer.parseInt(str)`
σε `long` → `Long.parseLong(str)`
σε `float` → `new Float(str).floatValue()`
σε `double` → `Double.parseDouble(str)`

Ο χαρακτήρας `\` χρησιμοποιείται σε ειδικές περιπτώσεις, όπως για την εκτύπωση των εισαγωγικών:

```
System.out.println("Say \"Hi\"!"); → τυπώνει: Say "Hi"!
```

(το παραπάνω string έχει 9 χαρακτήρες και όχι 11. Το `\` δε μετράει σαν χαρακτήρας)

Για εκτύπωση του `\`: `\\`

```
System.out.println("abc\\def");  → τυπώνει: abc\def
```

`\n` → νέα γραμμή (ENTER)
`\t` → tab

Η Κλάση `StringBuffer`

Είναι η κλάση για τη δημιουργία συμβολοσειρών που μπορούν να τροποποιηθούν.

Οι βασικοί της κατασκευαστές είναι οι εξής:

- `public StringBuffer()` - δημιουργία κενής συμβολοσειράς 16 θέσεων
- `public StringBuffer(String str)` - δημιουργία τροποποιήσιμης συμβολοσειράς με αρχική τιμή την τιμή `str`
- `public StringBuffer(int capacity)` - δημιουργία κενής συμβολοσειράς δεδομένου αριθμού θέσεων (capacity)

Στην ουσία τα αντικείμενα της `StringBuffer` είναι *διανύσματα χαρακτήρων* μεταβλητού μεγέθους.

Μια μέθοδος που χρησιμοποιείται συχνά για τροποποίηση μιας συμβολοσειράς (όχι δημιουργία καινούριας), είναι η `setCharAt`:

```
public void setCharAt(int index, char newChar)
```

η οποία αντικαθιστά τον υπάρχοντα χαρακτήρα στη θέση `index` μιας συμβολοσειράς με το χαρακτήρα `newChar`.

Η βασικότερη μέθοδος της κλάσης `StringBuffer` είναι η μέθοδος `append` η οποία υπάρχει σε πολλές *υπερφορτωμένες* μορφές. Η κύρια μορφή της είναι αυτή της συνένωσης συμβολοσειρών:

```
public void append(String str)
```

η οποία προσθέτει τη συμβολοσειρά `str` σε κάποιο συγκεκριμένο `StringBuffer`.

Π.χ., έστω ότι έχουν δηλωθεί και έχουν πάρει τιμές τρεις μεταβλητές τύπου `String`, οι: `title`, `firstName` και `lastName`. Εάν προσπαθούσε κάποιος να ενώσει αυτές τις τρεις συμβολοσειρές συνενώνοντας `String` και αποθηκεύοντας το αποτέλεσμα σε ένα νέο `String` με την εντολή:

```
String name = title + " " + firstName + " " + lastName;
```

τότε στην ουσία θα είχε δημιουργήσει 4 ενδιάμεσα `String` μέχρι να επιτευχθεί η τελική συμβολοσειρά. Αντιθέτως, με τη χρήση της μεθόδου `append` της `StringBuffer`, η συνένωση των συμβολοσειρών θα γινόταν με τροποποίηση μιας μόνο συμβολοσειράς:

```
StringBuffer name = new StringBuffer().append(title).append(" ").
                    append(firstName).append(" ").append(lastName);
```

Μετατροπές `String` / `StringBuffer`

Η μέθοδος `toString` μπορεί να μετατρέψει ένα `StringBuffer` σε `String`. Κάνοντας χρήση της ικανότητας της `StringBuffer` να τροποποιεί συμβολοσειρές και της `toString`, μπορούμε να δημιουργήσουμε μία μέθοδο που δέχεται κάποιο `String` και το επιστρέφει τροποποιημένο:

```
public String changeString(String str)
{
   StringBuffer buffer = new StringBuffer(str);

   // τροποποίηση του buffer
   .
   .
   .

   return buffer.toString();
}
```

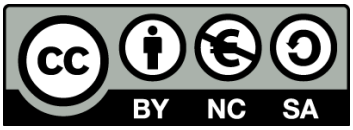
Στην ουσία η μέθοδος αυτή κάνει το εξής: δέχεται μια συμβολοσειρά `String`, την αποθηκεύει σε μορφή `StringBuffer`, την τροποποιεί στην επιθυμητή νέα τιμή και επιστρέφει την τροποποιημένη συμβολοσειρά σε μορφή `String` πάλι.

Σημειώματα

Σημείωμα Αναφοράς

Μιχάλης Δρακόπουλος. «Πληροφορική ΙΙ. Ενότητα 6: Pass-by-value και φαινομενικό pass-by-reference». 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.