ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ

Ενότητα 7: Πίνακες (Arrays)

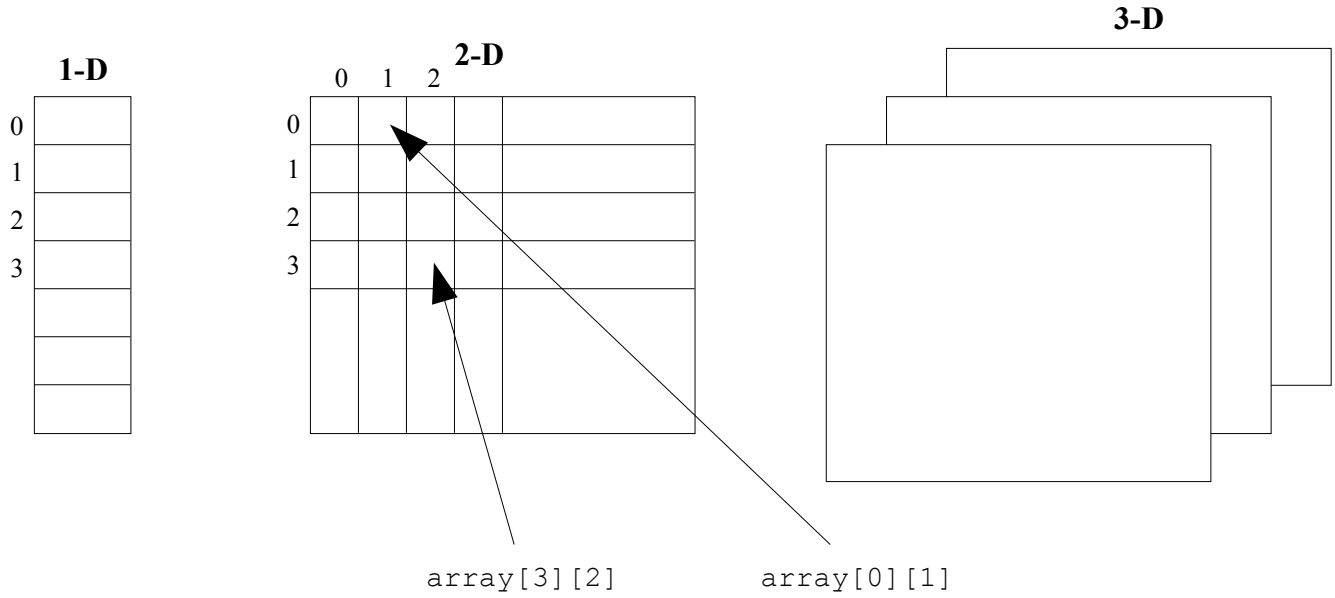
Μιχάλης Δρακόπουλος

Σχολή Θετικών επιστημών

Τμήμα Μαθηματικών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ (Java)
Ενότητα 7

Πίνακες (Arrays)



Γενική δήλωση και δημιουργία μονοδιάστατου πίνακα:

```
τύπος_στοιχείων [ ] όνομα = new τύπος_στοιχείων [μήκος];  // → 1-D
```

και φυσικά μπροστά μπορεί να υπάρχει η ανάλογη ορατότητα (`public`, `private`, κτλ.)

π.χ. `private double [ ] temperature = new double [10];`

Στην ουσία, με αυτόν τον τρόπο δηλώνονται 10 `double` μεταβλητές υπό το ίδιο όνομα, δηλαδή *σαν* να δηλώνονταν:

```
private double temperature[0] = 0;
private double temperature[1] = 0;
  ...
private double temperature[9] = 0;
```

ο τρόπος αυτός δήλωσης μεταβλητών όμως δεν υφίσταται πραγματικά, γι αυτό και η δήλωσή τους γίνεται μέσω της δημιουργίας του αντίστοιχου πίνακα `temperature`.

Σημαντικά σημεία εδώ είναι ότι:

- Η δημιουργία ενός πίνακα αρχικοποιεί τα στοιχεία του στην τιμή 0 (null, για στοιχεία τύπου char).
- Η αρίθμηση της θέσης των στοιχείων του πίνακα (index) ξεκινάει από την τιμή 0 (και άρα φτάνει μέχρι την τιμή «μήκος–1»)

Δήλωση και δημιουργία δισδιάστατου πίνακα (2-D):

```
τύπος_στοιχείων [ ][ ] όνομα = new τύπος_στοιχείων [size1][size2];
```

Δημιουργία και αρχικοποίηση σε συγκεκριμένες τιμές διαφορετικές του μηδενός:

```
τύπος_στοιχείων [ ] όνομα = {τιμή1, τιμή2, ...};
```

π.χ., `int [ ] daysInMonth = {31, 28, 31, 30, 31, ...};`

ή

```
int [ ] daysInMonth = new int [12];
daysInMonth[0] = 31;
daysInMonth[1] = 28;
daysInMonth[2] = 31;
  ...
daysInMonth[11] = 31;
```

ΠΡΟΣΟΧΗ: Το `daysInMonth[12]` **δεν** υπάρχει. 12 είναι το μέγεθος του πίνακα. Το index μεταβάλλεται από 0 έως 11.

Για 2-D:

π.χ. η εντολή:

```
int [ ][ ] a = { {2, 3, 5}, {4, 2, 0}, {0, 2, 1} };
```

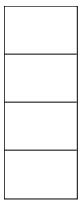
δημιουργεί τον πίνακα:

```
     2  3  5
a =  4  2  0
     0  2  1
```

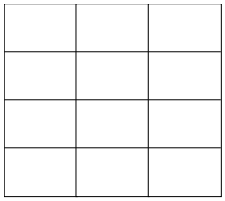
Λεπτομέρεια: Στη Java τυπικά υπάρχουν μόνο μονοδιάστατοι πίνακες! Όταν φτιάχνουμε έναν δισδιάστατο πίνακα, στην ουσία φτιάχνουμε ένα μονοδιάστατο πίνακα για κάθε στοιχείο ενός αρχικού μονοδιάστατο πίνακα. Δηλαδή, όταν φτιάχνουμε ένα δισδιάστατο πίνακα μεγέθους [4]x[3] με την εντολή:

```
int [ ][ ] array1 = new int [4][3];
```

στην αρχή δημιουργείται ένας 1-D πίνακας μεγέθους [4]:



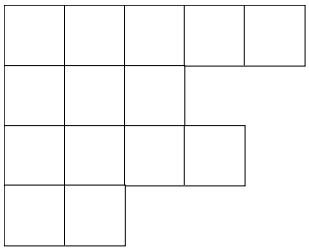
και στη συνέχεια κάθε στοιχείο του γίνεται ένας 1-D πίνακας μεγέθους [3], άρα φαινομενικά δημιουργείται ο 2-D πίνακας διαστάσεων [4] x [3]:



Αυτό δίνει τη δυνατότητα στη Java να δημιουργήσει πίνακες μη-ορθογώνιους, δηλαδή πίνακες με διαφορετικό αριθμό στηλών ανά γραμμή:

```
int [ ][ ] array2 = new int [4][ ];
array2[0] = new int [5];
array2[1] = new int [3];
array2[2] = new int [4];
array2[3] = new int [2];
```

Οι εντολές αυτές δημιουργούν τον ακόλουθο πίνακα:



Η `length` επιστρέφει το μέγεθος ενός πίνακα:

```
double [ ] a = new double [10];
int x = a.length;   // → x=10
```

Οπότε, ένας καλός τρόπος αυτοματοποιημένης αρχικοποίησης ενός πίνακα:
(παράδειγμα):

```
int [ ] b = new int [100];
for (int i=0; i<b.length; i++)
{
   b[i] = 1;
}
```

Στην περίπτωση 2-D πινάκων:

```
double [ ][ ] c = new double [10][20];
for (int i=0; i<c.length; i++)
{
   for (int j=0; j<c[0].length; j++)
   {
      c[i][j] = 1;
   }
}
```

Το `c[0].length` στην ουσία επιστρέφει το μήκος της πρώτης γραμμής του πίνακα (της γραμμής 0), δηλαδή 20, και το 0 στο `c[0]` θα μπορούσε να είναι οποιοσδήποτε ακέραιος μεταξύ 0 και 9 εφόσον ο `c` είναι ορθογώνιος πίνακας (όλες η γραμμές έχουν το ίδιο πλήθος στοιχείων).

ΕΡΩΤΗΣΗ: Πώς θα μπορούσαμε με παραπλήσιο τρόπο να αρχικοποιήσουμε *μη-ορθογώνιο* δισδιάστατο πίνακα;

Παράδειγμα

Να γραφεί μέθοδος που να δέχεται ακέραιο n, να δημιουργεί δισδιάστατο πίνακα (n x n) ακέραιων τιμών και να αρχικοποιεί το άνω τριγωνικό του μέρος με την τιμή i*j για κάθε στοιχείο (i,j).

```
public int [][] myMethod(int n)
{
   int [][] arr = new int [n][n];   // δε χρειάζεται αρχικοποίηση στο 0

   // μετατροπή του άνω τριγωνικού μέρους μόνο:
   for (int i=0; i<n; i++)
   {
      for (int j=i; j<n; j++)
      {
         arr[i][j] = i*j;
      }
   }
   return arr;
}
```

Σε περίπτωση που η παραπάνω μέθοδος ανήκε σε μια κλάση `SuppClass` και θέλαμε να την καλέσουμε από μια άλλη κλάση `AppClass`, θα είχαμε κάτι σαν το παρακάτω:

```
public class AppClass
{

   public static void main(String [] args)
   {
      // δήλωση ενός δισδιάστατου πίνακα ακεραίων (βλέπε 1η παρατήρηση παρακάτω)
      int [][] myArray;

         // δημιουργία αντικειμένου της κλάσης SuppClass
      SuppClass obj = new SuppClass();

         // κλήση της μεθόδου για μορφοποίηση του άνω τριγωνικού
         // μέρους του πίνακα
      myArray = obj.myMethod(10);

         // στο σημείο αυτό, ο πίνακας myArray έχει πάρει τιμές
         // όπως προβλέπει η μέθοδος myMethod της κλάσης SuppClass
   }
}
```

Παρατηρήσεις:

- Η εντολή δήλωσης και δημιουργίας ενός πίνακα που αναφέρθηκε στην αρχή της ενότητας, ουσιαστικά είναι δύο εντολές: η *δήλωση* του πίνακα (π.χ., `int [][] myArray;`) και η *δημιουργία* του με τη `new` (π.χ. `myArray = new int[10][10];`). Στο συγκεκριμένο παράδειγμα είναι περιττό να δημιουργηθεί ο πίνακας `myArray`, αφού στην ουσία δημιουργείται και παίρνει τιμές μέσα στη μέθοδο `myMethod`, η οποία τον επιστρέφει. Πρέπει όμως μέσα στη `main` να δηλωθεί ο συγκεκριμένος πίνακας, ώστε στη συνέχεια να αποθηκευτεί σε αυτόν ο πίνακας που επιστρέφεται από τη μέθοδο `myMethod`.
- Αν στην κλάση `AppClass` ο πίνακας `myArray` δηλωνόταν *εκτός* της μεθόδου `main` (δηλαδή ήταν πίνακας κλάσης και όχι τοπικός πίνακας της `main`), τότε θα έπρεπε να ορισθεί ως `static`, γιατί η `main` είναι `static` μέθοδος και `static` μέθοδοι έχουν πρόσβαση μόνο σε `static` στοιχεία (μεταβλητές και πίνακες).

Αντιγραφή πινάκων

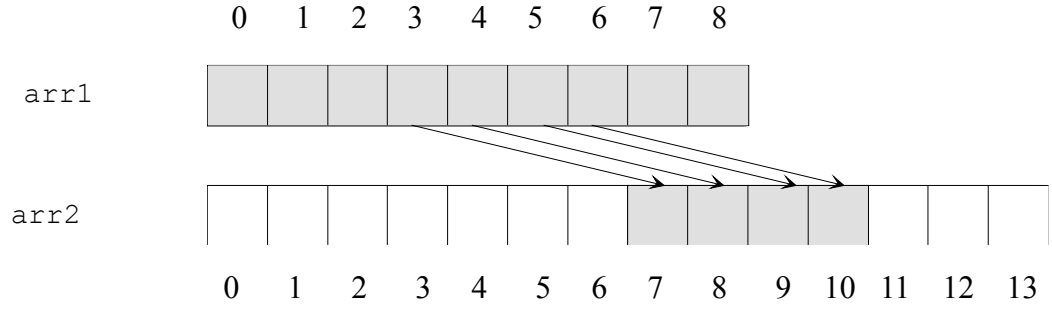
```
// δημιουργία δύο πινάκων
int [ ] array1 = new int [5];
int [ ] array2 = new int [5];

// αρχικοποίηση του πίνακα array1
for (int i=0; i<array1.length; i++)
{
   array1[i] = 2*i;   // array1 = {0,2,4,6,8}
}

// προσπάθεια αντιγραφής του πίνακα array1 στον πίνακα array2
array2 = array1;   // Ίδιος χώρος στη μνήμη. Ένας πίνακας με δύο ονόματα!

// Στην ουσία αυτό δεν έκανε αντιγραφή! Αν:
array2[0]=100;
System.out.println(array1[0]); // τυπώνει 100 και όχι 0
// δηλ., αλλάζοντας τον array2 αλλάζει και ο array1.
// Λογικό, αφού πρόκειται για ΕΝΑΝ πίνακα με δύο ονόματα...

// Ο σωστός τρόπος αντιγραφής του array1 στον array2:

System.arraycopy(array1, 0, array2, 0, array1.length);
```

Δηλαδή, για αντιγραφή πινάκων χρησιμοποιούμε τη `System.arraycopy`:

```
System.arraycopy(sourceArray, sourceArrayStartingPosition,
       destinationArray, destinationArrayStartingPosition,
       howManyElementsToCopy);
```

όπου:
`sourceArray`: ο αρχικός πίνακας
`sourceArrayStartingPosition`: από ποιο σημείο να αρχίσει να παίρνει στοιχεία για αντιγραφή
`destinationArray`: ο τελικός πίνακας
`destinationArrayStartingPosition`: από ποιο σημείο να αρχίσει να αντιγράφει στο νέο πίνακα
`howManyElementsToCopy`: πόσα στοιχεία να αντιγραφούν

Άρα μπορούμε να αντιγράψουμε και τμήματα πινάκων:



`System.arraycopy(arr1,3,arr2,7,4);`

Πίνακες αντικειμένων

Οι πίνακες, εκτός από στοιχεία πρωτογενών τύπων (`int`, `double`, `char` κτλ.), μπορούν να αποτελούνται και από **στοιχεία αντικείμενα**. Όπως μεταβλητές του ίδιου τύπου "ομαδοποιούνται" σε ένα πίνακα υπό ένα κοινό όνομα, έτσι και αντικείμενα μιας κλάσης μπορούν να ομαδοποιηθούν υπό ένα κοινό όνομα σε ένα πίνακα αντικειμένων. Η μόνη διαφορά με τους κοινούς πίνακες είναι ότι οι πίνακες αντικειμένων είναι τύπου κάποιας κλάσης (και όχι πρωτογενούς τύπου) και τα στοιχεία τους είναι αντικείμενα και όχι απλές μεταβλητές.

Αν έχουμε μια κλάση υποστήριξης `SuppClass`, υπάρχουν δύο βήματα για τη δημιουργία αντικειμένων της κλάσης αυτής που θα αποτελούν στοιχεία πίνακα:

i) δημιουργία πίνακα:

```
SuppClass [ ] objects = new SuppClass [5];
```

Έτσι δημιουργείται ο πίνακας `objects` με μέγεθος 5, του οποίου τα στοιχεία είναι τύπου `SuppClass`, δηλαδή αντικείμενα της κλάσης `SuppClass`. Άρα, ο πίνακας `objects` θα περιέχει 5 αντικείμενα της κλάσης `SuppClass`.

ii) δημιουργία αντικειμένων:

```
for (int i=0; i<objects.length; i++)
{
   objects[i] = new SuppClass();
}
```

Έτσι, δημιουργήθηκαν τα 5 αντικείμενα.

Οπότε, για κλήση π.χ. της μεθόδου `method1()` της `SuppClass` μέσω του αντικειμένου `objects[2]`:

```
objects[2].method1();
```

ή της `method2()` από κάποιο άλλο αντικείμενο, π.χ. το `objects[0]`:

```
objects[0].method2();
```

Μέθοδοι πρόσβασης που επιστρέφουν πίνακες:

```
...
// δημιουργία ενός private πίνακα:
private double [][] a = new double [5][10];
...

// μέθοδος πρόσβασης που επιστρέφει τον private πίνακα
public double [][] accessArray()
{
   return a;
}
```

Μέθοδοι που δέχονται πίνακες σαν παράμετρο εισόδου:

```
public int [] initArray(int [] arr)
{
   for (int i=0; i<arr.length; i++)
   {
      arr[i] = 1;
   }
   return arr;
}
```

Η συγκεκριμένη μέθοδος δέχεται έναν μονοδιάστατο πίνακα ακέραιων τιμών, αρχικοποιεί τα στοιχεία του στην τιμή 1 και τον επιστρέφει.

Όταν *δεν* ξέρουμε εκ των προτέρων τη διάσταση του πίνακα που θέλουμε να δημιουργήσουμε:

Π.χ., θέλουμε να έχουμε ένα μονοδιάστατο πίνακα ακέραιων στοιχείων στην κλάση μας, αλλά δε γνωρίζουμε εκ των προτέρων το μέγεθός του, δηλαδή πόσες `int` τιμές θα θέλουμε να κρατάει. Σε αυτή την περίπτωση, μπορούμε να χωρίσουμε στη μέση τη σύνθετη εντολή δήλωσης και δημιουργίας του πίνακα (όπως αναφέρθηκε και παραπάνω), έτσι ώστε το πρώτο της μέρος, δηλαδή η δήλωση του πίνακα, να είναι στην κλάση (πριν τις μεθόδους) και άρα ο πίνακας να ανήκει σε όλη την κλάση, και το δεύτερο μέρος, που είναι και η πραγματική δημιουργία του πίνακα, να είναι μέσα σε μία μέθοδο που να δέχεται σαν παράμετρο εισόδου το μέγεθος του πίνακα:

```
public class Example
{

  private int [] myArray;  // δήλωση του πίνακα κλάσης
    .
    .
    .
  public void setArray(int x)
  {
     myArray = new int [x]; // δημιουργία του πίνακα
  }

     .
     .
     .

  // αν μετά έχω μια μέθοδο που π.χ. αρχικοποιεί αυτόν τον πίνακα:
  public void initArray()
  {
     for (int i=0; i<myArray.length; i++)
        myArray[i] = 1;
  }

}
```

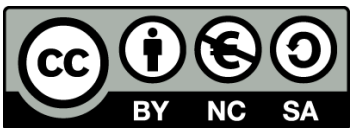
Θα πρέπει να προσέξω να καλέσω πρώτα τη `setArray` και μετά την `initArray`, για οποιοδήποτε αντικείμενο αυτής της κλάσης, διαφορετικά θα υπάρξει NullPointerException (ένα σφάλμα-εξαίρεση που μπορούμε να διαχειριστούμε, όπως θα δούμε αργότερα), γιατί ο πίνακας στην ουσία δεν έχει δημιουργηθεί πριν γίνει κλήση της `setArray`, άρα το `myArray.length` αλλά και η απόδοση τιμής σε στοιχεία του πίνακα κατά την κλήση της `initArray`, θα ήταν προβληματικά.

Σημειώματα

Σημείωμα Αναφοράς

Μιχάλης Δρακόπουλος. «Πληροφορική ΙΙ. Ενότητα 7: Πίνακες (Arrays)». 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.