ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ

Ενότητα 11: Vectors (διανύσματα)

Μιχάλης Δρακόπουλος

Σχολή Θετικών επιστημών

Τμήμα Μαθηματικών

ΠΛΗΡΟΦΟΡΙΚΗ ΙΙ (Java)
Ενότητα 11

Vectors (διανύσματα)

Τα vectors είναι δυναμικές δομές δεδομένων.
Κυριότερο χαρακτηριστικό τους: έχουν δυναμικό (μεταβαλλόμενο) μέγεθος.

Το μέγεθος ενός πίνακα δεν μπορεί να αλλάξει από τη στιγμή που ο πίνακας οριστεί, ακόμα και αν το μέγεθος δεν ορίζεται επ'ακριβώς μέσα στο πρόγραμμα αλλά το δίνει ο χρήστης σαν input στο πρόγραμμα.

π.χ.
```
private double [ ] a;
...
public void createArray(int size)
{
      a = new double [size];
}
```

δηλαδή αρχικά το μέγεθος του πίνακα `a` δεν είναι γνωστό, αλλά όταν κληθεί η μέθοδος createArray, το μέγεθος θα είναι γνωστό και θα σταλεί σαν όρισμα από τον χρήστη κατά την εντολή κλήσης της μεθόδου, οπότε ο πίνακας δημιουργείται μέσα στη μέθοδο (από τη στιγμή όμως που έχει ορισθεί έξω από τη μέθοδο, παραμένει ένας πίνακας κλάσης, δηλαδή η εμβέλειά του (scope) είναι ολόκληρη η κλάση).

Αν κατόπιν θέλουμε να έχουμε περισσότερα στοιχεία (elements) στον `a`, τότε υπάρχει πρόβλημα!
Για αυτό το λόγο υπάρχουν τα vectors, που έχουν μεταβλητό μέγεθος.

→ Τότε γιατί να μην χρησιμοποιούμε πάντα vectors αντί για πίνακες;
Τα vectors έχουν δύο βασικά μειονεκτήματα:

- Είναι λιγότερο αποτελεσματικά από τους πίνακες, δηλαδή πιο δύσκολα στη χρήση τους
- Τα στοιχεία τους πρέπει να είναι αντικείμενα (objects). Δεν μπορούν να είναι απλές τιμές πρωτογενών μεταβλητών (int, double, char, κτλ.) όπως στους πίνακες (αυτό φυσικά έχει και πάρα πολλά πλεονεκτήματα)

Χρήση των vectors:

Ο ορισμός της κλάσης `Vector` δεν παρέχεται αυτόματα. Βρίσκεται στο πακέτο `java.util` , το οποίο πρέπει να γίνει import από οποιοδήποτε πρόγραμμα χρησιμοποιεί vectors:

```
import java.util.*;
```

Δημιουργία vector:

Η `Vector` είναι κλάση, άρα τα vectors δημιουργούνται όπως δημιουργούνται τα αντικείμενα:

```
Vector vectorName = new Vector();
```

Κάθε vector έχει μια χωρητικότητα (capacity), μια «αύξηση χωρητικότητας» (capacityIncrement) και ένα μέγεθος (size). Η χωρητικότητά του είναι πάντα μεγαλύτερη ή ίση με το μέγεθός του (capacity >= size). Σαν μέγεθος θεωρούμε το πλήθος των στοιχείων που έχει ανά πάσα στιγμή το vector.

Κατασκευαστές της `Vector`:

- `public Vector()` – δημιουργεί ένα κενό vector με αρχική χωρητικότητα = 10. Κάθε φορά που χρειάζεται αύξηση της χωρητικότητας, η χωρητικότητα διπλασιάζεται.

- `public Vector(int initialCapacity)` – όπως προηγουμένως, αλλά η αρχική χωρητικότητα είναι = `initialCapacity`.

  π.χ., `Vector v1 = new Vector(35);`

- `public Vector(int initialCapacity, int capacityIncrement)` – όπως προηγουμένως, αλλά η χωρητικότητα μεγαλώνει κατά "`capacityIncrement`" στοιχεία κάθε φορά που χρειάζεται.

Μέθοδοι της `Vector`:

- **Προσθήκη στοιχείου:**

- στο τέλος του vector (μετά το τελευταίο στοιχείο)

```
public void addElement(Object newElement)
```

π.χ., αν έχουμε το vector `v2`:

4	2	1	0	15	9	

και θέλουμε να προσθέσουμε μια `int` τιμή στο τέλος του vector:

```
private int x = 2;
v2.addElement(new Integer(x));
```
→

4	2	1	0	15	9	2	

- ενδιάμεσα:

```
public void insertElementAt(Object newElement, int index)
```

π.χ., `v2.insertElementAt(new Integer(7),4);` →

4	2	1	0	7	15	9	2	

Σημείωση: Η τρέχουσα έκδοση της Java δεν απαιτεί πλέον τη δημιουργία αντικειμένου για την προσθήκη του ως στοιχείο του vector, αφού έχει αυτοματοποιήσει τη μετατροπή του σε αντικείμενο. Δηλαδή, μπορεί κάποιος να προσθέσει κατευθείαν έναν πρωτογενή τύπο σε διάνυσμα και σε αυτή την περίπτωση η Java θα κάνει αυτόματα τη μετατροπή του στο αντίστοιχο αντικείμενο. Άρα, π.χ., η προηγούμενη προσθήκη:

```
v2.insertElementAt(new Integer(7),4);
```

θα μπορούσε να γραφεί απλά ως:

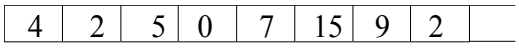
```
v2.insertElementAt(7,4);
```

- **Αντικατάσταση (υπάρχοντος) στοιχείου:**

```
public void setElementAt(Object newElement, int index)
```

π.χ., θέλουμε να αλλάξουμε το *τρίτο* στοιχείο του `v2` σε "5":

`v2.setElementAt(new Integer(5), 2);` →

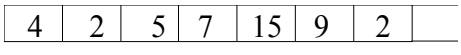


(ή, πιο απλά: `v2.setElementAt(5, 2);`)

- **Διαγραφή στοιχείου:**

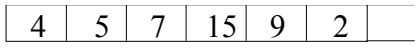
- `public void removeElementAt(int index)`

π.χ., `v2.removeElementAt(3);` →



- `public boolean removeElement(Object theElement)`
  (διαγράφει το στοιχείο theElement *μόνο από την πρώτη του θέση* στο vector – επιστρέφει true εάν βρει το στοιχείο και το διαγράψει, και false εάν δεν το βρει)

π.χ., `v2.removeElement(new Integer(2));` → `true` →



`v2.removeElement(new Integer(3));` → `false`

- `public void removeAllElements()`

- **Πρόσβαση σε στοιχείο:**

- `public Object elementAt(int index)`

π.χ., θέλουμε να προσθέσουμε τις τιμές του 3ου και 5ου στοιχείου του v2:

```
int sum = (Integer)v2.elementAt(2) + (Integer)v2.elementAt(4);
          |         |____Object_____| |
          |_____Integer object _________|
```

Σημείωση: Η Java σε αυτή την περίπτωση έχει αυτοματοποιήσει ακόμα μία διαδικασία, αυτή της μετατροπής του αντικειμένου Integer στον αντίστοιχο πρωτογενή τύπο (int). Κάτι τέτοιο σε παλαιότερες εκδόσεις της Java γίνονταν μέσω εξειδικευμένων μεθόδων. Για να γίνει όμως η αυτόματη μετατροπή σε int, πρέπει το αντικείμενο Object που επιστρέφει η elementAt να μετατραπεί σε αντικείμενο Integer. Χωρίς αυτή τη μετατροπή, η Java δεν μπορεί να μετατρέψει τα στοιχεία σε πρωτογενείς τύπους, άρα η έκφραση:

```
int sum = v2.elementAt(2) + v2.elementAt(4);   // ΛΑΘΟΣ!
```

θα έβγαζε σφάλμα κατά το compilation.

- <u>Μέθοδοι αναζήτησης:</u>

- `public boolean contains(Object target)` → `true/false`

π.χ., αν έχουμε το vector v2 στη μορφή:

4	2	5	7	15	9	2	

`v2.contains(new Integer(15));` → `true`

- `public int indexOf(Object target) -> 1ο index του target ή -1 αν δεν υπάρχει`

π.χ., πάντα για το ίδιο v2:

`v2.indexOf(new Integer(2));` → `1`

`v2.indexOf(new Integer(3));` → `-1`

- `public int indexOf(Object target, int StartIndex)`

π.χ., `v2.indexOf(new Integer(2), 3);` → `6`

- `public int lastIndexOf(Object target)`

π.χ., `v2.lastIndexOf(new Integer(2));` → `6`

- `public Object firstElement()`

π.χ., `v2.firstElement();` → `4` (το οποίο όμως είναι αντικείμενο!)

- `public Object lastElement()`

π.χ., `v3.lastElement();` → `2` (αντικείμενο)

- <u>Άλλες μέθοδοι:</u>

- `public int size()`

- `public int capacity()`

- `public void trimToSize()` → `κάνει το capacity = με το size`

- `public void setSize(int newSize)` → `κάνει το` <u>`μέγεθος`</u> `(και όχι το capacity) = με το newSize.`

Αν το `newSize` > προηγούμενο `size` → τα νέα στοιχεία = `null`.
Αν το `newSize` < προηγούμενο `size` → τα επιπλέον στοιχεία χάνονται.

π.χ., `Vector v3 = new Vector();`

`v3.addElement(new Double(2.3));` →

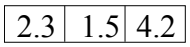
0	1	2	3	...	9
2.3				...	

`v3.addElement(new Double(1.5));` →

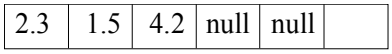
2.3	1.5			...	

```
v3.addElement(new Double(4.2)); →
```


```
int a = v3.size();  // → a = 3

int b = v3.capacity();  // → b = 10

v3.trimToSize();  →
```

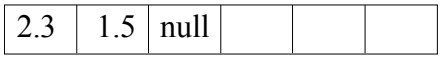



```
int c = v3.capacity(); // → c = 3 (το capacity γίνεται = με το size)

v3.setSize(5);  →
```




```
int d = v3.size()  // → d = 5

int e = v3.capacity();  // → e = 6  (το capacity διπλασιάστηκε)

v3.setSize(2);  →
```


```
int f = v3.capacity();  // → f = 6 (το capacity δεν άλλαξε)

v3.setSize(3);  →
```



(δηλαδή, η τιμή του 3ου στοιχείου (4.2) έχει χαθεί πλέον και δεν επανέρχεται)

Σημείωση: Υπάρχει διαφορά μεταξύ θέσεων με τιμή null και κενών θέσεων. Οι κενές θέσεις ενός vector στην ουσία δεν υπάρχουν μέχρι τη στιγμή που παίρνουν κάποια τιμή (ή την τιμή null). Υπάρχει μόνο η πρόβλεψη στη μνήμη για μελλοντική κάλυψη αυτών των θέσεων. *Αυτό* το νόημα έχει το capacity. Αν π.χ. κάποιος προσπαθήσει να προσπελάσει μια θέση ενός vector που «υπάρχει» στην χωρητικότητά του αλλά δεν έχει κάποια τιμή ή την τιμή null, ο compiler θα βγάλει σφάλμα (εξαίρεση): java.lang.ArrayIndexOutOfBoundsException

Παράδειγμα χρήσης vector:

- Να γραφεί μέθοδος που να ζητάει από τον χρήστη εισαγωγή double τιμών διαφορετικών του μηδενός μέχρι να δοθεί η τιμή 0 (για τερματισμό της διαδικασίας), και να αποθηκεύει τις τιμές αυτές.

Μπορούμε να χρησιμοποιήσουμε έναν double πίνακα;
Όχι, γιατί δεν ξέρουμε εξ αρχής πόσες τιμές θα εισάγει ο χρήστης! Άρα, χρησιμοποιούμε vector για την αποθήκευση των τιμών. Ένας τρόπος είναι ο ακόλουθος (υπάρχουν φυσικά διάφοροι τρόποι για να έχουμε το ίδιο αποτέλεσμα....):

```
import java.util.*;    // για χρήση του Vector

public class VectorExample
{
   Vector data = new Vector(); // δημιουργία του Vector "data"
   Scanner input = new Scanner(System.in);  // για user input

   // Η μέθοδος που ζητείται:
```

```
   public void askAndStoreData()
   {
      double d;
      do
      {
          System.out.println("Δώσε έναν πραγματικό αριθμό. Δώσε 0 για τερματισμό.");

         d = input.nextDouble();

         data.addElement(new Double(d));
            // ή: data.addElement(d);
      }
      while (d != 0);  // επανέλαβε όσο η τιμή δεν γίνεται 0

      // πρέπει να σβήσουμε το 0 που αποθηκεύτηκε στο τέλος!

      data.removeElement(0.0);

      // ή: data.removeElementAt(data.size()-1);
      // ή: data.setSize(data.size()-1);

   } // end method

} // end class
```

Προσοχή χρειάζεται στο σημείο που αφαιρούμε το στοιχείο 0.0 από το τέλος του vector. Αν γράψουμε τη σύντομη μορφή της εντολής όπως φαίνεται παραπάνω αντί της: `data.removeElement(new Double(0))`, θα πρέπει να γράψουμε 0.0 και όχι απλά 0, γιατί στη δεύτερη περίπτωση η java μετατρέπει αυτόματα το 0 σε `Integer` και όχι σε `Double`, οπότε δεν βρίσκει το αντικείμενο `Double 0` που βρίσκεται στο τέλος του vector! Προφανώς, οι δύο εναλλακτικοί τρόποι διαγραφής του μηδενός που φαίνονται στα σχόλια του κώδικα είναι προτιμότεροι, αφού δεν κάνουν αναζήτηση και άρα είναι πιο γρήγοροι.

Σημείωση: Αναφέρθηκε ότι ο μόνος «περιορισμός» είναι ότι τα στοιχεία των vectors πρέπει να είναι αντικείμενα. Άρα, ένα vector μπορεί να περιέχει (φαινομενικά τουλάχιστον) στοιχεία διαφορετικών τύπων (φαινομενικά γιατί στην ουσία όλα τα στοιχεία είναι αντικείμενα). Π.χ., ένα vector μπορεί να είναι το εξής:

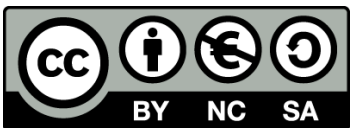
index	**vector**	
0	3.14	--> double
1	10	--> int
2	Hello	--> String
3	a	--> char
4	5 7 2	--> πίνακας

Σημειώματα

Σημείωμα Αναφοράς

Μιχάλης Δρακόπουλος. «Πληροφορική ΙΙ. Ενότητα 11: Vectors (διανύσματα)». 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.