2 ΓΡΑΜΜΙΚΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

(Linear Data Structures)

Ας θεωρήσουμε μία δομή δεδομένων το σύνολο των στοιχείων της οποίας είναι διατεταγμένο με τέτοιο τρόπο ώστε να ισχύουν τα εξής:

- **(α)** υπάρχει ένα μόνο στοιχείο το οποίο ονομάζεται αρχή και έχει ένα και μόνον ένα επόμενο στοιχείο
- **(β)** υπάρχει ένα μόνο στοιχείο το οποίο ονομάζεται τέλος και έχει ένα και μόνον ένα προηγούμενο
- **(γ)** κάθε άλλο στοιχείο έχει ένα και μόνον ένα προηγούμενο και ένα και μόνον ένα επόμενο

Στην περίπτωση αυτή λέμε ότι τα στοιχεία του συνόλου που αποτελούν τη δομή δεδομένων είναι γραμμικά διατεταγμένα και η δομή ονομάζεται **γραμμική δομή δεδομένων (linear data structure)**.

Στο κεφάλαιο αυτό εξετάζονται οι γραμμικές δομές δεδομένων πίνακας (array), το διάνυσμα (vector) και η συμβολοσειρά (string). Γραμμικές δομές δεδομένων είναι επίσης η στοίβα και η ουρά που εξετάζονται στο κεφάλαιο 3, καθώς και η συνδεδεμένη λίστα που παρουσιάζεται στο κεφάλαιο 4.

2.1 Πίνακες (*Arrays*)

Οι πίνακες στη Java θεωρούνται αντικείμενα και επομένως οι μεταβλητές τύπου πίνακα συμπεριφέρονται σαν μεταβλητές τύπου κλάσης, είναι δηλαδή αναφορές (references) σε αντικείμενα. Αν και δεν υπάρχει κλάση με την ονομασία **Array** αυτή υλοποιείται εσωτερικά από τη Java για λόγους αποτελεσματικότητας.

Είναι σημαντικό να διαχωρίζουμε τη μεταβλητή πίνακα και το στιγμιότυπο του πίνακα στον οποίο αναφέρεται κατά το χρόνο εκτέλεσης του προγράμματος. Δηλώνοντας ένα πίνακα όπως παρακάτω απλά δηλώνουμε μία μεταβλητή πίνακα τα στοχεία του οποίου πρέπει να είναι ακέραιοι αριθμοί:

```
int[ ] pin;
```

Ισοδύναμα η παραπάνω δήλωση χρησιμοποιώντας το συντακτικό της C/C++ μπορεί να γραφεί και:

```
int pin[ ];
```

Για να δημιουργήσουμε ένα στιγμιότυπο του πίνακα πρέπει να χρησιμοποιήσουμε τον τελεστή **new** καθορίζοντας σε συνδυασμό και με τον τελεστή **[ ]** το πλήθος των στοιχείων του πίνακα, όπως για παράδειγμα:

```
pin = new int[10];
```

Με την παραπάνω δήλωση γίνεται και ανάθεση μνήμης (στην περιοχή Heap) για τα στοιχεία του πίνακα. Το μέγεθος του πίνακα μπορεί να καθοριστεί από οποιαδήποτε έκφραση ακέραιου τύπου και δεν μπορεί να αλλάζει κατά τη διάρκεια της εκτέλεσης του προγράμματος:

```
int j = 5;
int max = 10 *j;
int [ ] pin = new int[max];
```

Ισοδύναμα δήλωση μεταβλητής και ανάθεση μνήμης για τα στοιχεία του πίνακα μπορεί να γίνει ταυτόχρονα:

```
int[ ] pin = new int[10];
```

Πολλές φορές είναι χρήσιμο να δώσουμε αρχικές τιμές στα στοιχεία ενός πίνακα, κάτι που αναφέρεται σαν αρχικοποίηση πίνακα (array initialization). Με την παρακάτω πρόταση:

```
int[ ] pin = {2,4,6,8,10,12,14,16,18,20};
```

γίνεται ταυτόχρονα δήλωση της μεταβλητής **pin**, ανάθεση μνήμης και καθορισμός του μεγέθους του πίνακα και τοποθετούνται αρχικές τιμές στα δέκα στοιχεία του πίνακα. Αξίζει να παρατηρήσουμε στην παραπάνω πρόταση την με έμμεσο τρόπο - χωρίς τη χρήση του τελεστή new - δημιουργία ενός στιγμιότυπου της κλάσης πίνακα. Κάτι τέτοιο επιτρέπεται αποκλειστικά λόγω της εσωτερικής υλοποίησης του πίνακα από τη Java.

Πρέπει να ξεχωρίζουμε την έννοια της αρχικοποίησης των στοιχείων του πίνακα από αυτήν της αρχικοποίησης μεταβλητής πίνακα. Με την παρακάτω πρόταση:

```
char[ ] data = null;
```

δηλώνουμε μία μεταβλητή πίνακα χαρακτήρων **data** η οποία έχει αρχική τιμή null. Η δυνατότητα αρχικοποίησης μεταβλητής πίνακα πηγάζει από το γεγονός ότι αυτή είναι μία μεταβλητή αναφοράς.

Τα στοιχεία ενός πίνακα μπορεί να είναι οποιοδήποτε τύπου, είτε βασικού τύπου είτε τύπου κλάσης. Στη δεύτερη περίπτωση αντιμετωπίζουμε κάθε στοιχείο του πίνακα σαν μία μεταβλητή τύπου κλάσης.

Επίσης τα στοιχεία ενός πίνακα μπορεί να έχουν τύπο που ορίζεται από κλάση διασύνδεσης (interface). Στην περίπτωση αυτή τα στοιχεία μπορούν να έχουν σαν τιμή τους αναφορές σε στιγμιότυπα οποιασδήποτε κλάσης υλοποιεί τη διασύνδεση ή την τιμή **null**.

Τέλος τα στοιχεία ενός πίνακα μπορεί να έχουν τύπο που ορίζεται από αφηρημένη (abstract) κλάση. Στην περίπτωση αυτή τα στοιχεία μπορούν να έχουν σαν τιμή τους αναφορές σε στιγμιότυπα οποιασδήποτε υπο-κλάσης της αφηρημένης κλάσης που όμως η ίδια δεν είναι αφηρημένη.

Στην περίπτωση που τα στοιχεία του πίνακα δεν αρχικοποιηθούν άμεσα με τη χρήση δήλωσης αυτό γίνεται έμμεσα όπως φαίνεται στον πίνακα 2.1.

Τύπος στοιχείων	Αρχικοποίηση
ακέραιος ή πραγματικός	0
χαρακτήρας	ο χαρακτήρας null
boolean	false
αντικείμενα τύπου κλάσης	null

Πίνακας 2.1

Για να αναφερθούμε σε ένα στοιχείο του πίνακα πρέπει να χρησιμοποιήσουμε το όνομα της μεταβλητής του πίνακα ακολουθούμενο από τον τελεστή **[ ]** περιλαμβάνοντας ένα συγκεκριμένο ενδείκτη (subscript). Ο ενδείκτης του πίνακα πρέπει να είναι φυσικός αριθμός ή έκφραση:

```
a = 3;
b = 5;
pin[a+b] += 2;
```

Η Java πραγματοποιεί έλεγχο εύρους (range checking) στους ενδείκτες των πινάκων κατά τη διάρκεια της εκτέλεσης του προγράμματος. Προσπάθεια να χρησιμοποιήσουμε ενδείκτη με τιμή μικρότερη του μηδενός ή μεγαλύτερη ή ίση του μήκους του πίνακα οδηγεί στην κατάσταση εξαίρεσης **IndexOutOfBoundExeption**.

Το μήκος του πίνακα (πλήθος των στοιχείων του) μπορεί να επιστραφεί με τη χρήση του μέλους δεδομένων (data member) **length** που αντιστοιχεί σε κάθε πίνακα:

```
int i = pin.length
```

Το μήκος του πίνακα δεν μπορεί να αλλάζει δυναμικά κατά τη διάρκεια εκτέλεσης του προγράμματος καθώς η τιμή του **length** είναι τιμή μόνον ανάγνωσης:

```
pin.length =100; // Προσοχή, λάθος!
```

Τα στοιχεία ενός πίνακα μπορεί να είναι με τη σειρά τους αλλοι πίνακες. Στην περίπτωση αυτή ο πίνακας ονομάζεται πολυδιάστατος. Παραδείγματα πολυδιάστατων πινάκων δίνονται παρακάτω:

```
int[ ] [ ] pin2;
```

```
pin2 = new int [3] [4];

int[ ] [ ]  pin3= { {4,5}, {1,7} };
```

Τα στοιχεία ενός πολυδιάστατου πίνακα μπορεί να είναι πίνακες διαφορετικού μήκους όπως φαίνεται στο επόμενο παράδειγμα:

```
int[ ] [ ] pin4;
pin4 = new int[2] [ ];
pin4[0] = new int[3];
pin4[1] = new int[5];
```

όπου ο πίνακας **pin4** αποτελείται από δύο γραμμές: ένα πίνακα μήκους 3 και ένα πίνακα μήκους 5.

2.2 Διανύσματα (Vectors)

Ενώ ένας πίνακας είναι στατικός ως προς το μέγεθός του, με τη βοήθεια της κλάσης **Vector** μπορούμε να υλοποιήσουμε διανύσματα - μία μορφή πινάκων οι οποίοι μπορούν να μεταβάλουν το μήκος τους κατά τη διάρκεια της εκτέλεσης του προγράμματος. Ένα διάνυσμα περιέχει στοιχεία τα οποία μπορεί να είναι οποιοδήποτε τύπου και τα στοιχεία αυτά είναι προσβάσιμα όπως και στην περίπτωση του πίνακα μέσω ενός ακέραιου ενδείκτη. Η κλάση **Vector** ορίζεται στα πλαίσια του πακέτου **java.util**.

Μπορούμε να δημιουργήσουμε ένα διάνυσμα με τρεις διαφορετικούς τρόπους:

(α) Καθορίζοντας την αρχική του χωρητικότητα. Στο παράδειγμα που ακολουθεί το διάνυσμα v1 έχει αρχική χωρητικότητα μία θέση.

```
Vector v1 = new Vector(1);
```

Κάθε φορά που παρουσιάζεται ανάγκη χρήσης περισσότερων στοιχείων η χωρητικότητα του διανύσματος διπλασιάζεται.

(β) Να μην ορίσουμε ρητά την αρχική χωρητικότητα του διανύσματος, όπως φαίνεται στο παρακάτω παράδειγμα:

```
Vector v2 = new Vector( );
```

Στην περίπτωση αυτή η αρχική χωρητικότητα καθορίζεται από το σύστημα της Java να είναι 10 θέσεις. Και στην περίπτωση αυτή κάθε φορά που παρουσιάζεται ανάγκη χρήσης περισσότερων στοιχείων η χωρητικότητα του διανύσματος διπλασιάζεται.

(γ) Να ορίσουμε την αρχική χωρητικότητα του διανύσματος, καθώς και το βήμα αύξησης της χωρητικότητας, όπως φαίνεται στο παρακάτω παράδειγμα:

```
Vector v3 = new Vector(10, 5);
```

Στην περίπτωση αυτή η αρχική χωρητικότητα είναι 10 θέσεις. Αυξάνεται κατά 5 όταν υπάρχει ανάγκη

Οι βασικές λειτουργίες που αναφέρονται στα διανύσματα υλοποιούνται από τις μεθόδους που φαίνονται στον πίνακα 2.2.

Μέθοδος	Περιγραφή
void addElement(Object Item)	προσθέτει το Item στο τέλος του διανύσματος αυξάνοντας το μέγεθός του κατά 1.
Void insertElementAt(Object Item, int pos)	Παρεμβάλει το Item σαν νέο στοιχείο του διανύσματος στη θέση pos
void setElementAt(Object Item, int pos)	Τοποθετεί στο υπ' αριθμό pos στοιχείο του διανύσματος το Item
boolean removeElement(Object Item)	Διαγράφει την πρώτη εμφάνιση του στοιχείου Item από το διάνυσμα
boolean isEmpty	Ελέγχει εάν το διάνυσμα δεν έχει καθόλου στοιχεία
Object firstElement( )	Επιστρέφει το πρώτο στοιχείο του διανύσματος (αυτό που βρίσκεται στη θέση 0)

Πίνακας 2.2

Μέθοδος	Περιγραφή
Object lastElement( )	Επιστρέφει το τελευταίο στοιχείο του διανύσματος
boolean contains(Object Item)	Ελέγχει εάν το Item αποτελεί στοιχείο του διανύσματος
void trimToSize( )	"Ψαλιδίζει" τη χωρητικότητα του διανύσματος ώστε να γίνει ίση με το τρέχον μέγεθός του
int size( )	Επιστρέφει το τρέχον μέγεθος του διανύσματος
int capacity( )	Επιστρέφει τη χωρητικότητα του διανύσματος

Πίνακας 2.2 (συνέχεια)

2.3 Συμβολοσειρές (Strings)

Στη Java υπάρχουν δύο κλάσεις με τη βοήθεια των οποίων μπορούμε να δημιουργήσουμε και να χειριστούμε συμβολοσειρές. Η κλάση **String** και η κλάση **StringBuffer** οι οποίες ορίζονται στα πλαίσια του πακέτου **java.lang**. Τα στιγμιότυπα και της κλάσης **String** και της κλάσης **StringBuffer** αναπαριστούν ακολουθίες Unicode χαρακτήρων.

Οι συμβολοσειρές που δημιουργούνται στα πλαίσια της κλάσης **String** ονομάζονται **αμετάβλητες (immutable)**. Ένα αντικείμενο τύπου **String** από τη στιγμή που θα δημιουργηθεί δεν μπορεί να αλλάξει τιμή. Αυτό σημαίνει ότι οποιαδήποτε λειτουργία τροποποιεί μία συμβολοσειρά οδηγεί στη δημιουργία ενός νέου αντικειμένου.

Έστω ότι δύνονται δύο συμβολοσειράς **S** και **Q**. Στον πίνακα 2.3 περιγράφονται οι μέθοδοι που αντιστοιχούν στις βασικές πράξεις του τύπου **String**.

Μέθοδος	Περιγραφή
length( )	Επιστρέφει το μήκος της συμβολοσειράς **S**
charAt(i)	Επιστρέφει τον χαρακτήρα που βρίσκεται στη θέση **i** της συμβολοσειράς **S**

Πίνακας 2.3

Μέθοδος	Περιγραφή
concat(Q)	Επιστρέφει τη συνένωση των συμβολοσειρών **S** και **Q** χωρίς να αλλάζει την τιμή της **S**
endsWith(Q)	Ελέγχει εάν το string **S** τελειώνει με το string **Q**
equals(Q)	Ελέγχει εάν το string **S** είναι ίσο με το string **Q**
indexOf(Q)	Εάν το string **Q** είναι υπό-string του **S** και επιστρέφει τον ενδείκτη της πρώτης εμφάνισης του στο **S**, αλλιώς επιστρέφει -1
startsWith(Q)	
substring(i,j)	Επιστρέφει το υπό-string του **S** που αρχίζει από τη θέση **i** και τελειώνει στη θέση **j**

Πίνακας 2.3 (συνέχεια)

Με το πρόγραμμα που φαίνεται στο τμήμα κώδικα 1 μπορεί να ελεγχθεί η λειτουργία των βασικών πράξεων του τύπου **String**.

```
class CheckString {

   public static void main(String arguments[ ]) {
      String str = "Πολύ μου αρέσει η γλώσσα προγραμματισμού Java";
      System.out.println("Η συμβολοσειρά είναι: " + str);
      System.out.println("Το μήκος της συμβολοσειράς είναι: " + str.length( ));
      System.out.println("Ο χαρακτήρας στη θέση 7: " + str.charAt(7));
      System.out.println("Το υπο-string από 24 μέχρι 31:" + str.substring(24,31));
      System.out.println("Ο ενδείκτης του χαρακτήρα x: " + str.indexOf('x'));
      System.out.println("Ο ενδείκτης της αρχής του " + "υπο-string \"γλώσσα\": "
                        + str.indexOf("γλώσσα"));
      System.out.println("Το string με κεφαλαία: " + str.toUpperCase());
   }
}
```

Τμήμα κώδικα 1

Οι συμβολοσειρές που δημιουργούνται στα πλαίσια της κλάσης **StringBuffer** ονομάζονται **ευμετάβλητες (mutable)**. Ένα αντικείμενο τύπου **StringBuffer** μπορεί να αλλάζει τιμή κατά τη διάρκεια της εκτέλεσης του προγράμματος

Δοθείσης μίας συμβολοσειράς **S** και μίας **Q** στον παρακάτω πίνακα (πίνακας 2.4) περιγράφονται οι μέθοδοι που αντιστοιχούν στις βασικές πράξεις του τύπου **StringBuffer**.

Μέθοδος	Περιγραφή
append(Q)	Επιστρέφει τη συνένωση των συμβολοσειρών **S** και **Q** επιστρέφοντας το αποτέλεσμα στη συμβολοσειρά **S**.
insert(i,Q)	Επιστρέφει τη συμβολοσειρά που προκύπτει από την εισαγωγή της **Q** στη συμβολοσειρά **S** ξεκινώντας από τη θέση **i** της **S**. Το αποτέλεσμα τροποποιεί την **S**.
reverse( )	Αντιστρέφει τη συμβολοσειρά **S**
setCharAt(i,ch)	Τοποθετεί τον υπ' αριθμό **i** χαρακτήρα της συμβολοσειράς **S** με **ch**. Το αποτέλεσμα τροποποιεί την **S**.

Πίνακας 2.4

2.4 Η κλάση StringTokenizer

Όταν επεξεργαζόμαστε μία μεγάλη συμβολοσειρά, όπως για παράδειγμα ένα κομμάτι κειμένου τότε είναι λογικό να τη χωρίζουμε σε επί μέρους τμήματα - λέξεις, τα οποία ανάλογα με τη φύση του κειμένου έχουν κάποιο συγκεκριμένο νόημα. Τα τμήματα αυτά ονομάζονται **tokens**. Τα tokens ξεχωρίζουν μεταξύ τους με τη βοήθεια ειδικών οριοθετών. Οι οριοθέτες αυτοί μπορεί να είναι:

- ένας ή περισσότεροι κενοί χαρακτήρες
- ειδικοί χαρακτήρες, όπως **tab** και **newline**
- ειδικοί χαρακτήρες αλλαγής γραμμής, όπως **newline** και **carriage return**

Στη Java στα πλαίσια του πακέτου **java.util** ορίζεται η κλάση **StringTokenizer**, η οποία είναι ιδιαίτερα χρήσιμη για τον χωρισμό μίας συμβολοσειράς σε tokens και τον χειρισμό των tokens αυτών.

Στα τμήματα κώδικα 2 και 3 φαίνονται δύο παραλλαγές ενός προγράμματος, όπου χρησιμοποιείται η κλάση **StringTokenizer**.

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

public class Setences
{

public static void main(String[ ] args) throws IOException
{
      String s1;
      int i;
      BufferedReader br;
      br = new BufferedReader(new InputStreamReader(System.in));

      s1 = br.readLine( );
      StringTokenizer tokens = new StringTokenizer(s1);

      System.out.println(tokens.countTokens( ));
      while (tokens.hasMoreTokens( ))
          System.out.println(tokens.nextToken( ));
}
}
```

Τμήμα Κώδικα 2

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

public class Setences2
{

public static void main(String[ ] args) throws IOException
{
      String s1;
      int i,j;
      String ArrayOfTokens[ ] = new String[20];

      BufferedReader mybuffer;
      mybuffer = new BufferedReader(new InputStreamReader(System.in));

      s1 = mybuffer.readLine();
      StringTokenizer tokens = new StringTokenizer(s1);

      i = tokens.countTokens();
      System.out.println(i);
      j = 0;
      while (tokens.hasMoreTokens( ) && j < ArrayOfTokens.length)
      {    ArrayOfTokens[j] = tokens.nextToken( );
           j++;
      }
     for (j=0; j<i; j++)
     {
        System.out.println(ArrayOfTokens[j]);
     }
}
}
```

Τμήμα κώδικα 3