6 ΑΡΧΕΙΑ και ΡΕΥΜΑΤΑ (Files & Streams)

6.1 Βασικές Έννοιες

Ένα **αρχείο *(file)*** είναι μία σύνθετη δομή δεδομένων που αποτελείται από μία σειρά **στοιχείων *(elements)***, τα οποία είναι (συνήθως) του ίδιου τύπου. Ο αριθμός αυτών των στοιχείων δεν είναι περιορισμένος και μπορεί να μεταβάλλεται δυναμικά κατά τη διάρκεια της ζωής του αρχείου. Ο μεταβλητός αυτός αριθμός στοιχείων του αρχείου οριοθετείται από ένα ειδικό σύμβολο ή τιμή που ονομάζεται τέλος αρχείου (end-of-file) και συνήθως συμβολίζεται με EOF (σχήμα 6.1). Ο αριθμός των στοιχείων ενός αρχείου (σε κάποια χρονική στιγμή) ονομάζεται μήκος του αρχείου.

Στοιχείο 1	Στοιχείο 2	. . . . .	Στοιχείο N	EOF

Σχήμα 6.1: Λογική αναπαράσταση ενός αρχείου

Τα στοιχεία ενός αρχείου ονομάζονται και **λογικές εγγραφές** ή απλά **εγγραφές *(structures)*.** Κάθε στοιχείο ή εγγραφή του αρχείου συναθροίζει (συνήθως) ένα σύνολο διαφορετικών επιμέρους τύπων δεδομένων που από κοινού περιγράφουν κάποια οντότητα. Οι επί μέρους αυτοί τύποι δεδομένων ονομάζονται **πεδία *(fields)*** της εγγραφής.

Σαν παράδειγμα μπορούμε να θεωρήσουμε ένα αρχείο το οποίο περιλαμβάνει τους συνδρομητές μίας τηλεφωνικής εταιρείας. Για κάθε συνδρομητή πρέπει να καταχωρίζουμε το ονοματεπώνυμο, τη διεύθυνση κατοικίας, το επάγγελμα και τον

αριθμό τηλεφώνου. Στην περίπτωση αυτή κάθε συνδρομητής αποτελεί ένα στοιχείο (ή εγγραφή) του αρχείου. Οι επί μέρους πληροφορίες που περιγράφουν την οντότητα συνδρομητής αποτελούν τα πεδία της εγγραφής. Ένα πεδίο μπορεί να είναι είτε απλό είτε σύνθετο. Ένα απλό πεδίο (π.χ. το τηλέφωνο του συνδρομητή) περιγράφεται με ένα πρωταρχικό τύπο δεδομένων ενώ ένα σύνθετο πεδίο μπορεί να αποτελείται από επί μέρους πεδία. Στο παράδειγμα το πεδίο ονοματεπώνυμο μπορεί να είναι σύνθετο και να αποτελείται από το επίθετο, το όνομα και το πατρώνυμο του συνδρομητή.

Εξωτερικά αρχεία (αρχεία που είναι "γνωστά" στο λειτουργικό σύστημα) μπορούν να χρησιμοποιηθούν για την αποθήκευση δεδομένων των οποίων ο χρόνος ζωής είναι ανεξάρτητος από τη διάρκεια εκτέλεσης των προγραμμάτων. Στα πλαίσια μιας γλώσσας προγραμματισμού είναι δυνατός ο ορισμός και η επεξεργασία διαφόρων τύπων αρχείων. Οι πιο βασικοί τύποι αρχείων που μπορούν να οριστούν είναι τα **σειριακά** ή **ακολουθιακά αρχεία** *(sequential flies)* και τα αρχεία **κατ' ευθείαν πρόσβασης** *(direct access)*.

6.2 Αρχεία και Ρεύματα Bytes στη Java

Βασική προϋπόθεση για να γίνει κατανοητός ο τρόπος με τον οποίο η Java αντιμετωπίζει την επεξεργασία των αρχείων είναι η κατανόηση της έννοιας του ρεύματος.

Ρεύμα (*Stream*) είναι ένα αντικείμενο το οποίο αναπαριστά μία σειριακή ροή δεδομένων από μία πηγή προς έναν προορισμό.

Η Java αντιμετωπίζει ένα αρχείο σαν ένα ακολουθιακό ρεύμα (stream) από bytes. Κάθε αρχείο είτε τελειώνει με το ειδικό σύμβολο "end-of-file" είτε με ένα byte που περιέχει έναν συγκεκριμένο αριθμό που καθορίζεται από το σύστημα.

Όταν ένα αρχείο ανοίγει, ένα αντικείμενο δημιουργείται και ένα ρεύμα συσχετίζεται με αυτό.

Για να μεταφέρει δεδομένα ένα πρόγραμμα από ένα αρχείο προς την κεντρική μνήμη πρέπει να δημιουργήσει ένα ρεύμα εισόδου (input stream) και να διαβάσει τα δεδομένα του ακολουθιακά όπως φαίνεται στην παρακάτω διαδικασία:

Διαδικασία Ανάγνωσης:
ΒΗΜΑ 1: Δημιούργησε ένα ρεύμα εισόδου με βάση το αρχείο;
ΒΗΜΑ 2: Εφόσον <υπάρχουν δεδομένα>
Διάβασε επόμενο δεδομένο;
[Επεξεργάσου το δεδομένο;]
ΒΗΜΑ 3: Κλείσε το ρεύμα εισόδου;

Το ρεύμα εισόδου στην περίπτωση αυτή λειτουργεί σαν «μεσάζοντας». Το πρόγραμμα από τη στιγμή που θα δημιουργηθεί το ρεύμα εισόδου απευθύνεται προς αυτό και «ξεχνάει» το αρχείο.

Με όμοιο τρόπο, για να μεταφέρει δεδομένα ένα πρόγραμμα από την κεντρική μνήμη προς ένα αρχείο πρέπει να δημιουργήσει ένα ρεύμα εξόδου (output stream) και να γράψει τα δεδομένα του ακολουθιακά όπως φαίνεται στην παρακάτω διαδικασία:

Διαδικασία Εγγραφής:
ΒΗΜΑ 1: Δημιούργησε ένα ρεύμα εξόδου με βάση το αρχείο;
ΒΗΜΑ 2: Εφόσον <υπάρχουν δεδομένα>
[Προετοίμασε επόμενο δεδομένο;]
Γράψε δεδομένο;
ΒΗΜΑ 3: Κλείσε το ρεύμα εξόδου;

Το ρεύμα εξόδου και στην περίπτωση αυτή λειτουργεί σαν «μεσάζοντας». Το πρόγραμμα από τη στιγμή που θα δημιουργηθεί το ρεύμα εξόδου στέλνει τα δεδομένα του προς αυτό και «αγνοεί» την ύπαρξη του αρχείου.

Υπάρχει ένας μεγάλος αριθμός κλάσεων και ένας αριθμός από διασυνδέσεις (interfaces) στη Java ο οποίος χρησιμοποιείται για τον ορισμό και την επεξεργασία ρευμάτων και αρχείων. Οι κλάσεις και οι διασυνδέσεις αυτές είναι ορισμένες στο πακέτο java.io. Είναι πολύ σημαντικό να κατανοήσει κανείς την ιεραρχία αυτών των κλάσεων για να μπορέσει να τις χρησιμοποιήσει αποτελεσματικά για τον προγραμματισμό διαδικασιών εισόδου/εξόδου. Στο σχήμα 6.2 παρουσιάζονται κάποιες από τις κλάσεις αυτές, ιεραρχικά τοποθετημένες, που κρίνονται σαν βασικές, ενώ στο

Παράρτημα 1 φαίνονται όλες οι κλάσεις και διασυνδέσεις του πακέτου. Επειδή είναι προφανές ότι η αναλυτική περιγραφή όλων των μεθόδων που υλοποιούνται στα πλαίσια των κλάσεων αυτών δεν είναι δυνατόν να δοθεί εδώ, ο αναγνώστης παραπέμπεται στο API specification της γλώσσας στον επίσημο δικτυακό τόπο της Java, στο πακέτο java.io:
https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html
Εκεί μπορεί να δει την αναλυτική περιγραφή κάθε κλάσης που καλείται να χρησιμοποιήσει, καθώς και την αυστηρή περιγραφή κάθε μεθόδου που υλοποιεί κάποια διαδικασία εισόδου ή εξόδου. Στο σχήμα 6.3 φαίνεται η περιγραφή μίας τέτοιας μεθόδου.

Οι κλάσεις **FileInputStream**, **FileOutputStream**, **FileReader** και **FileWriter** υλοποιούν τις λειτουργίες των ρευμάτων αρχείων (file streams) και χρησιμοποιούνται για την είσοδο και έξοδο δεδομένων στα εξωτερικά αρχεία του συστήματος.

Οι **FileInputStream** και **FileOutputStream** χρησιμοποιούνται για ανάγνωση και εγγραφή Bytes (8 bits) και οι **FileReader** και **FileWriter** χρησιμοποιούνται για ανάγνωση και εγγραφή χαρακτήρων 16-bits (Unicode)

Κάποιες από τις κλάσεις που υλοποιούν ρεύματα χρησιμοποιούνται για «φιλτράρισμα» (filtering). Ένα **ρεύμα φίλτρο** φιλτράρει δεδομένα όπως αυτά διαβάζονται ή γράφονται στο ρεύμα. Στην περίπτωση αυτή το ρεύμα φίλτρο παρέχει επιπλέον λειτουργίες όπως είναι το buffering, η καταμέτρηση γραμμών ή η ομαδοποίηση δεδομένων. Παραδείγματα κλάσεων που υλοποιούν φίλτρα είναι η **FilterInputStream** και **FilterOutputStream** καθώς και οι υποκλάσεις τους **DataInputStream** και **DataOutputStream**.

Το **buffering** είναι μία ειδική περίπτωση υλοποίησης φίλτρων και αποτελεί μία τεχνική βελτίωσης της αποδοτικότητας των διαδικασιών εισόδου/εξόδου (I/O). Παραδείγματα κλάσεων που υλοποιούν buffering είναι η **BufferedInputStream** και **BufferedOutputStream**.

Η διασύνδεση **DataInput** ορίζει τις πράξεις ανάγνωσης όλων των βασικών τύπων της Java από ένα ρεύμα από bytes (byte stream). Με την έννοια αυτή ορίζει τον τρόπο που μία σειρά από bytes που διαβάζονται από ένα ρεύμα εισόδου δημιουργούν την τιμή

του τύπου που πρέπει να διαβαστεί. Επίσης ορίζει πράξεις ανάγνωσης Strings από ένα ρεύμα από bytes.

- InputStream (abstract class)
  - FileInputStream
  - FilterInputStream
    - DataInputStream (υλοποιεί το interface DataInput)
    - BufferedInputStream
    - LineNumberInputStream
  - ObjectInputStream

- OutputStream (abstract class)
  - FileOutputStream
  - FilterOutputStream
    - DataOutputStream (υλοποιεί το interface DataOutput)
    - BufferedOutputStream
  - ObjectOutputStream

- Reader (abstract class)
  - BufferedReader
  - InputSteamReader
    - FileReader

- Writer (abstract class)
  - BufferedWriter
  - OutputSteamWriter
    - FileWriter

- RandomAccessFile (υλοποιεί τα interfaces DataInput και DataOutput)

- File

ΣΧΗΜΑ 6.2:

Ιεραρχία κλάσεων για την επεξεργασία αρχείων (package java.io)

Η διασύνδεση **DataOutput** ορίζει τις πράξεις εγγραφής όλων των βασικών τύπων της Java σε ένα ρεύμα από bytes. Με την έννοια αυτή ορίζει τον τρόπο μετατροπής οποιασδήποτε τιμής ενός τύπου σε μία σειρά από bytes που πρέπει να γραφούν σε ένα ρεύμα εξόδου. Επίσης ορίζει πράξεις εγγραφής Strings σε ένα ρεύμα από bytes.

read

public int **read(**byte[ ] b**)** throws IOException

Διαβάζει έως b.length bytes δεδομένων από το ρεύμα εισόδου (input stream) και τους τοποθετεί σε έναν πίνακα από bytes. Η μέθοδος αυτή μπαίνει σε κατάσταση αναμονής (blocks) μέχρις ότου υπάρξουν διαθέσιμα δεδομένα.

Παράμετροι (Parameters):

b – ο πίνακας στον οποίον τοποθετούνται τα δεδομένα που διαβάζονται.

Επιστρέφει (Returns):

Τον ολικό αριθμό των Bytes που διαβάστηκαν στον πίνακα ή –1 αν δεν υπάρχουν άλλα δεδομένα γιατί η διαδικασία ανάγνωσης έφτασε στο τέλος του αρχείου (end of the file).

Προκαλεί (Throws):

IOException – εάν προκύψει λάθος εισόδου/εξόδου (I/O error).

Επαναορίζει (Overrides):

Τη **read** της κλάσης InputStream

ΣΧΗΜΑ 6.3: ***Παράδειγμα περιγραφής μεθόδου της FileInputStream όπως δίνεται από το API Specification της JAVA***

6.3 Ακολουθιακά Αρχεία (*sequential files*)

Ενα ακολουθιακό αρχείο μοιάζει, ως προς τη δομή του, με έναν πίνακα, αποτελείται δηλαδή από στοιχεία που είναι όλα του ίδιου τύπου, διαφέρει όμως από τον πίνακα σε δύο βασικά σημεία:

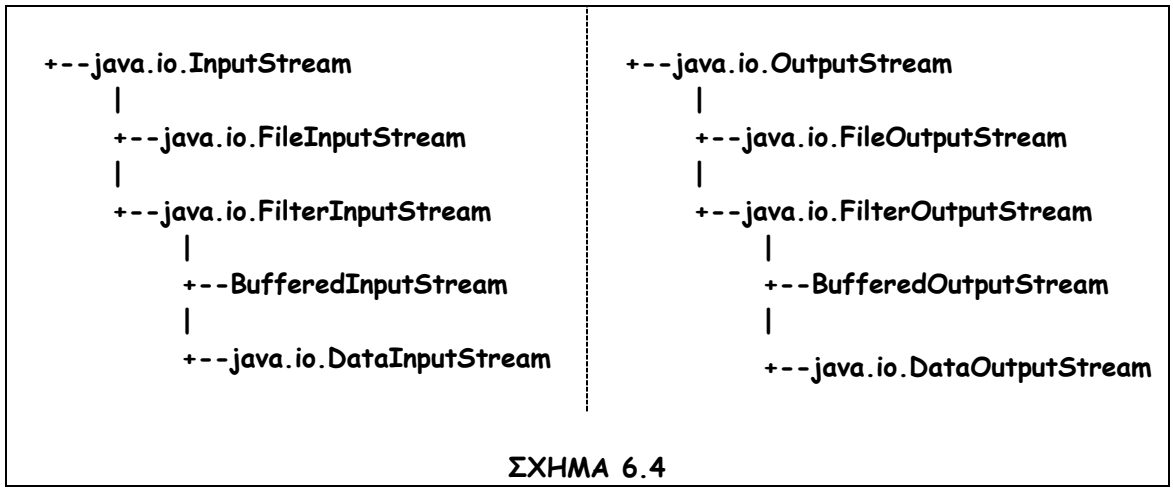
(α) το μέγεθος του: Το μέγεθος του πίνακα είναι προκαθορισμένο από το χρόνο μεταγλώττισης. Αντίθετα ένα ακολουθιακό αρχείο μπορεί να είναι αρχικά κενό (να περιέχει 0 στοιχεία), και να επεκτείνεται καθώς νέα στοιχεία προσθέτονται στη διάρκεια της εκτέλεσης του προγράμματος, που το χειρίζεται.

(β) τον τρόπο πρόσβασης: Τα στοιχεία ενός πίνακα είναι προσπελάσιμα με οποιαδήποτε σειρά (αρκεί να αναφέρουμε τον αριθμό του στοιχείου). Αντίθετα τα στοιχεία ενός ακολουθιακού αρχείου είναι προσπελάσιμα μόνο σειριακά. Για να προσπελάσουμε το Ν-στό στοιχείο του αρχείου, πρέπει να ξεκινήσουμε από το πρώτο και να "περάσουμε" από όλα τα προηγούμενα Ν-1 στοιχεία. (Παρατηρήστε την ομοιότητα πρόσβασης στα στοιχεία ενός ακολουθιακού αρχείου, με την πρόσβαση στη μουσική πληροφορία μιας κασέτας.)

Τα ακολουθιακά αρχεία κατά τη διάρκεια της επεξεργασίας τους ανοίγουν κατ' αποκλειστικότητα είτε σαν **αρχεία εισόδου** είτε σαν **αρχεία εξόδου**. Αυτό σημαίνει κυρίως ότι κατά τη διάρκεια της επεξεργασίας ενός αρχείου εισόδου δεν έχουμε τη δυνατότητα να τροποποιήσουμε τα δεδομένα του, καθώς δεν έχουμε δικαίωμα να εκτελέσουμε οποιαδήποτε διαδικασία εγγραφής σε αυτό.

6.3.1 Επεξεργασία Ακολουθιακών Αρχείων από Bytes

Η Java θεωρεί κατ' αρχήν ότι τα ρεύματα και τα αρχεία αποτελούνται από bytes. Τα bytes αυτά μπορούν στη συνέχεια να αντιμετωπιστούν ότι εν σειρά ορίζουν πιο σύνθετους τύπους δεδομένων. Ολες οι διαδικασίες εισόδου για ακολουθιακά ρεύματα και ακολουθιακά αρχεία υλοποιούνται μέσω των υποκλάσεων της αφηρημένης κλάσης **InputStream**, και όλες οι διαδικασίες εξόδου μέσω των υποκλάσεων της αφηρημένης κλάσης **OutputStream**. Οι πλέον απαραίτητες υποκλάσεις τους καθώς και ο τρόπος που αυτές σχετίζονται ιεραρχικά φαίνονται φαίνονται στο παρακάτω σχήμα:



ΣΧΗΜΑ 6.4

Οι βασικές διαδικασίες εισόδου εξόδου που ορίζονται στα πλαίσια των κλάσεων **InputStream** και **OutputStream** δίνονται στους πίνακες που ακολουθούν:

Βασικές Μέθοδοι της κλάσης InputStream

Μέθοδος	Περιγραφή
int read( )	Επιστρέφει έναν ακέραιο αριθμό που αντιστοιχεί στο επόμενο διαθέσιμο byte του stream εισόδου. Επιστρέφει –1 όταν συναντήσει το τέλος του αρχείου.
int read(byte buffer[ ])	Προσπαθεί να διαβάσει από το stream εισόδου αριθμό από bytes ίσο με το μήκος **buffer.length** του πίνακα **buffer** και τους τοποθετεί σε αυτόν. Επιστρέφει έναν ακέραιο αριθμό που αντιστοιχεί στον αριθμό των bytes που διάβασε ή. -1 άν συναντήσει το τέλος του αρχείου.
int available( )	Επιστρέφει τον αριθμό των bytes του stream εισόδου που είναι διαθέσιμα.
long skip(long numBytes)	Προσπαθεί να παρακάμψει τα επόμενα **numBytes** bytes του stream εισόδου επιστρέφοντας τον αριθμό όσων αγνόησε.
void close( )	Κλείνει το stream εισόδου. Μετά το κλείσιμο προσπάθειες ανάγνωσης δεδομένων οδηγούν σε εξαίρεση **IOException**

Βασικές Μέθοδοι της κλάσης OutputStream

Μέθοδος	Περιγραφή
void write(int oneByte)	Γράφει ένα μόνο byte στο stream εξόδου.
void write(byte buffer[ ])	Γράφει τα bytes που είναι αποθηκευμένα στον πίνακα **buffer** στο stream εξόδου.
void close( )	Κλείνει το stream εξόδου. Μετά το κλείσιμο προσπάθειες εγγραφής δεδομένων οδηγούν σε εξαίρεση **IOException**

Στο τμήμα κώδικα 1 δίνεται ένα πρόγραμμα που χειρίζεται ένα αρχείο εισόδου μέσω της **FileInputStream** και στο τμήμα κώδικα 2 ένα πρόγραμμα που χειρίζεται ένα αρχείο εξόδου μέσω της **FileOutputStream**.

```
import java.io.*;

public class ReadBytes {
   public static void main(String[] arguments) {
      try {
         FileInputStream file = new FileInputStream("class.dat");
         boolean eof = false;
         int count = 0;
         while (!eof) {
            int input = file.read();
            System.out.print(input + " ");
            if (input == -1)
               eof = true;
            else
               count++;
         }
         file.close();
         System.out.println("\nBytes read: " + count);
      } catch (IOException e) {
         System.out.println("Error -- " + e.toString());
      }
   }
}
```

Τμήμα Κώδικα 1

```
import java.io.*;

public class WriteBytes {
   public static void main(String[] arguments) {
      int[ ] data = { 71, 73, 70, 56, 57, 97, 15, 0, 15, 0,
         128, 0, 0, 255, 255, 255, 0, 0, 0, 44, 0, 0, 0,
         0, 15, 0, 15, 0, 0, 2, 33, 132, 127, 161, 200,
         185, 205, 84, 128, 241, 81, 35, 175, 155, 26,
         228, 254, 105, 33, 102, 121, 165, 201, 145, 169,
         154, 142, 172, 116, 162, 240, 90, 197, 5, 0, 59 };
      try {
         FileOutputStream file = new FileOutputStream("pic.gif");
         for (int i = 0; i < data.length; i++)
            file.write(data[i]);
         file.close();
      } catch (IOException e) {
         System.out.println("Error -- " + e.toString());
      }
   }
}
```

Τμήμα Κώδικα 2

Ένα ρεύμα **FilterInputStream** περιέχει ή δέχεται σαν είσοδο κάποιο άλλο ρεύμα εισόδου, το οποίο χρησιμοποιεί σαν πηγή δεδομένων και κατά κάποιο τρόπο του τροποποιεί τα δεδομένα ή παρέχει κάποιες επιπλέον λειτουργίες πάνω σ' αυτά. Η κλάση **FilterInputStream** επαναορίζει τις μεθόδους της **InputStream**. Οι υποκλάσεις της **FilterInputStream** μπορούν επίσης να επαναορίσουν κάποιες από αυτές τις μεθόδους ή να προσθέσουν νέες. Με αντίστοιχο τρόπο λειτουργούν και τα ρεύματα που ορίζονται από την κλάση **FilterOutputStream** και τις υποκλάσεις της σε σχέση με τα ρεύματα εξόδου. Στην επόμενη παράγραφο περιγράφονται οι υποκλάσεις των **FilterInputStream** και **FilterOutputStream** που χρησιμοποιούνται σαν φίλτρα σε ρεύματα εισόδου και εξόδου αντίστοιχα για την επεξεργασία των πρωταρχικών τύπων δεδομένων της Java.

6.3.2 Επεξεργασία Πρωταρχικών τύπων Δεδομένων

Οι κλάσεις **DataInputStream** και **DataOutputStream**

(**public class DataInputStream extends FilterInputStream implements DataInput**) και
(**public class DataOutputStream extends FilterOutputStream implements DataOutput**)

υλοποιούν τις λειτουργίες που σχετίζονται με την ανάγνωση και εγγραφή δεδομένων που αντιστοιχούν στους πρωταρχικούς τύπους της Java (primitive data types). Οι κλάσεις αυτές φροντίζουν στην ουσία για τη μετατροπή των δεδομένων που βρίσκονται με την μορφή bytes στα δεδομένα που αντιστοιχούν σε βασικούς τύπους. Για τον κάθε πρωταρχικό τύπο ορίζονται και οι κατάλληλες μέθοδοι για την ανάγνωση και εγγραφή του από ένα ρεύμα εισόδου και σε ένα ρεύμα εξόδου αντίστοιχα. Οι διαδικασίες αυτές ανάγνωσης/εγγραφής είναι ανεξάρτητες της μηχανής στην οποία τρέχει ένα πρόγραμμα της Java. Στους παρακάτω δύο πίνακες δίνονται ενδεικτικά μερικές από τις μεθόδους αυτές.

Ενδεικτικές Μέθοδοι της κλάσης DataInputStream

Μέθοδος	Περιγραφή
boolean readBoolean( )	Διαβάζει το επόμενο byte από το ρεύμα εισόδου. Επιστρέφει true εάν η τιμή του είναι μη μηδεκική και false εάν είναι μηδέν.
int readInt( )	Διαβάζει τα επόμενα τέσσερα bytes από το ρεύμα εισόδου και επιστρέφει τον ακέραιο αριθμό τύπου int που αντιστοιχεί σε αυτά.
float readFloat( )	Διαβάζει τα επόμενα τέσσερα bytes από το ρεύμα εισόδου και επιστρέφει τον πραγματικό αριθμό τύπου float που αντιστοιχεί σε αυτά.

Ενδεικτικές Μέθοδοι της κλάσης DataOutputStream

Μέθοδος	Περιγραφή
void writeInt(int n)	Γράφει έναν ακέραιο αριθμό στο ρεύμα εξόδου σαν μία σειρά από τέσσερα bytes, αρχίζοντας από το περισσότερο σημαντικό byte.
void writeFloat(float n)	Μετατρέπει τον πραγματικό αριθμό n σε ακέραιο χρησιμοποιώντας τη μέθοδο **floatToIntBits της κλάσης Float**, και στη συνέχεια γράφει τον αριθμό αυτό στο ρεύμα εξόδου σαν μία σειρά από τέσσερα bytes, αρχίζοντας από το περισσότερο σημαντικό.
void close( )	Κλείνει το stream εξόδου. Μετά το κλείσιμο προσπάθειες εγγραφής δεδομένων οδηγούν σε εξαίρεση **IOException**

Στο τμήμα κώδικα 3 δίνεται ένα πρόγραμμα που χειρίζεται ένα αρχείο εισόδου από το οποίο διαβάζει ακεραίους αριθμούς.

Μέσω της **FileInputStream** δημιουργείται ένα ρεύμα εισόδου στο οποίο διεχετεύονται τα δεδομένα του αρχείου. Η έξοδος του ρεύματος αυτού διοδεύεται σε έναν buffer ο οποίος δημιουργείται μέσω της **BufferedInputStream**. Ο ρόλος του buffer αυτού είναι να αυξήσει την ταχύτητα ροής των δεδομένων καθώς αυτά δεν τοποθετούνται στο ρεύμα ένα-ένα, αλλά κατά ομάδες (όσο είναι το μέγεθος του buffer). Στη συνέχεια τα δεδομένα του buffer διοδεύονται σε ένα ρεύμα-φίλτρο που δημιουργείται μέσω της **DataInputStream**. Το ρεύμα αυτό είναι υπεύθυνο να μετατρέψει μία σειρά από bytes (4 στη συγκεκριμένη περίπτωση) σε έναν ακέραιο κάθε φορά που εκτελείται η μέθοδος **readInt()**. Αντίστοιχα στο τμήμα κώδικα 4 δίνεται ένα πρόγραμμα που χειρίζεται ένα αρχείο εξόδου μέσω των κλάσεων **FileOutputStream**, **BufferedOutputStream** και **DataOutputStream**, στο οποίο εγγράφονται ακέραιοι αριθμοί.

```
import java.io.*;

class ReadPrimes {
   public static void main(String arguments[]) {
      try {
         FileInputStream file = new FileInputStream("400primes.dat");
         BufferedInputStream buff = new BufferedInputStream(file);
         DataInputStream data = new DataInputStream(buff);

         try {
            while (true) {
               int in = data.readInt();
               System.out.print(in + " ");
            }
         } catch (EOFException eof) {
            buff.close();
         }
      } catch (IOException e) {
         System.out.println("Error -- " + e.toString());
      }
   }
}
```

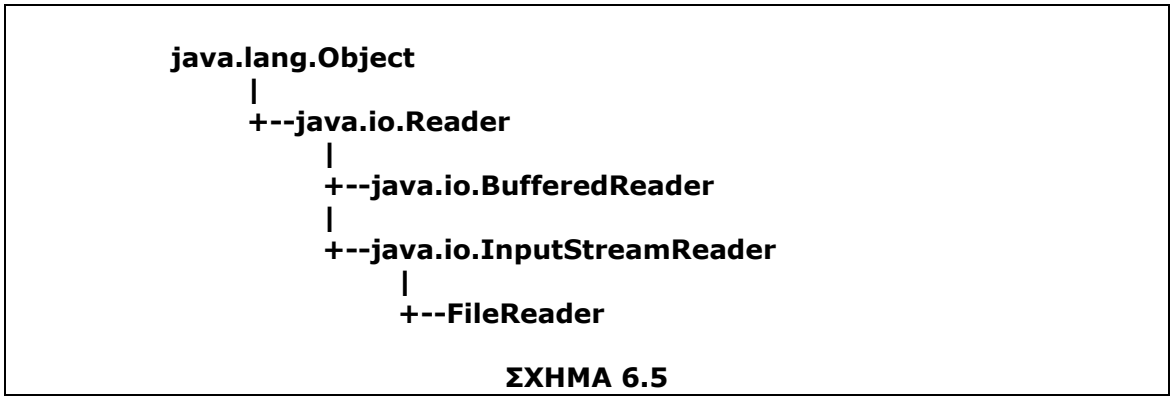
Τμήμα Κώδικα 3

```
import java.io.*;

class WritePrimes {
   public static void main(String arguments[]) {
      int[] primes = new int[400];
      int numPrimes = 0;
      // candidate: the number that might be prime
      int candidate = 2;
      while (numPrimes < 400) {
         if (isPrime(candidate)) {
            primes[numPrimes] = candidate;
            numPrimes++;
         }
         candidate++;
      }

      try {
         // Write output to disk
         FileOutputStream file = new FileOutputStream("400primes.dat");
         BufferedOutputStream buff = new BufferedOutputStream(file);
         DataOutputStream data = new DataOutputStream(buff);

         for (int i = 0; i < 400; i++)
            data.writeInt(primes[i]);
         data.close();
      } catch (IOException e) {
         System.out.println("Error -- " + e.toString());
      }
   }
```

Τμήμα Κώδικα 4

6.3.3 Αρχεία και Ρεύματα Χαρακτήρων – Αρχεία Κειμένου (text files)

Οι βασικές κλάσεις με τις οποίες μπορεί να χειριστεί κανείς αρχεία εισόδου χαρακτήρων στη Java φαίνονται παρακάτω:



ΣΧΗΜΑ 6.5

Η κλάση **Reader** ***(public abstract class Reader extends Object)*** είναι μία αφηρημένη κλάση που υλοποιεί τις διαδικασίες εισόδου από ρεύματα χαρακτήρων. Οι υποκλάσεις της κατ' αρχήν επαναυλοποιούν τις μεθόδους **read(char[], int, int)** και **close()**. Επίσης χρειάζεται να επαναυλοποιήσουν και κάποιες άλλες για λόγους αποτελεσματικότητας και λειτουργικότητας.

Βασικές Μέθοδοι της κλάσης Reader

Μέθοδος	Περιγραφή
int read( )	Διαβάζει έναν χαρακτήρα και τον επιστρέφει σαν έναν ακέραιο από 0 ως 16383. Επιστρέφει –1 εάν συναντήσει το τέλος του ρεύματος.
int read(char[ ] cbuf)	Διαβάζει χαρακτήρες και τους τοποθετεί στον πίνακα **cbuf**. Επιστρέφει τον αριθμό των bytes που διάβασε ή –1 εάν συναντήσει το τέλος του ρεύματος.
long skip(long n)	Προσπαθεί να παρακάμψει τους επόμενους n χαρακτήρες του stream εισόδου. Επιστρέφει τον αριθμό των χαρακτήρων που κατάφερε να παρακάμψει.
abstract void close()	Κλείνει το stream εισόδου.

Η κλάση **BufferedReader** ***(public class BufferedReader extends Reader)*** χρησιμοποιείται σαν φίλτρο για την ανάγνωση κειμένου από ένα ρεύμα χαρακτήρων με τη μέθοδο του buffering για λόγους αποτελεσματικότητας. Το μέγεθος του buffer μπορεί να οριστεί ή να χρησιμοποιηθεί η εξ' ορισμού τιμή του που συνήθως αρκεί για τις περισσότερες προγραμματιστικές εφαρμογές.

Βασικές Μέθοδοι της κλάσης BufferedReader

Μέθοδος	Περιγραφή
public int read( )	Διαβάζει έναν χαρακτήρα Επαναορίζει την read της κλάσης Reader
public String readLine( )	Διαβάζει μία γραμμή κειμένου. Μία γραμμή θεωρείται ότι τελειώνει είτε από τον χαρακτήρα line feed ('\n') είτε από τον χαρακτήρα *carriage return* ('\r'), ή από τους *carriage return* και *linefeed*. Επιστρέφει ένα String με τους χαρακτήρες της γραμμής στο οποίο δεν συμπεριλαμβάνονται οι ειδικοί χαρακτήρες αλλαγής γραμμής. Επιστρέφει null αν συναντήσει
long skip(long numChars)	Προσπαθεί να παρακάμψει τους επόμενους **numChars** χαρακτήρες του stream εισόδου επιστρέφοντας τον αριθμό όσων αγνόησε.
void close( )	Κλείνει το stream εισόδου. Μετά το κλείσιμο προσπάθειες ανάγνωσης δεδομένων οδηγούν σε εξαίρεση **IOException**

Ενα ρεύμα που δημιουργείται με την κλάση **InputStreamReader** ***(public class InputStreamReader extends Reader)*** αποτελεί μία «γέφυρα» από ένα ρεύμα Bytes σε ένα ρεύμα χαρακτήρων: Διαβάζει bytes από το ρεύμα εισόδου και τα μετατρέπει σε χαρακτήρες με βάση μία κωδικοποίηση χαρακτήρων. Η κωδικοποίηση αυτή μπορεί να είναι κατ' αρχήν αυτή που προσφέρει η Java (Unicodes), μπορεί όμως, αν αυτό είναι απαραίτητο, να οριστεί διαφορετικά.

Μία κλήση σε κάποια από τις μεθόδους read() της **InputStreamReader** προκαλεί την ανάγνωση ενός ή περισσοτέρων bytes από το ρεύμα των bytes που παρέχει την είσοδο.

Για λόγους αποτελεσματικότητας συνήθως ένα ρεύμα της **InputStreamReader** φιλτράρεται από ένα ρεύμα της **BufferedReader**, όπως για παράδειγμα:

```
InputStreamReader Isr = new InputStreamReader(System.in);
BufferedReader bufIsr = new BufferedReader(Isr);
```

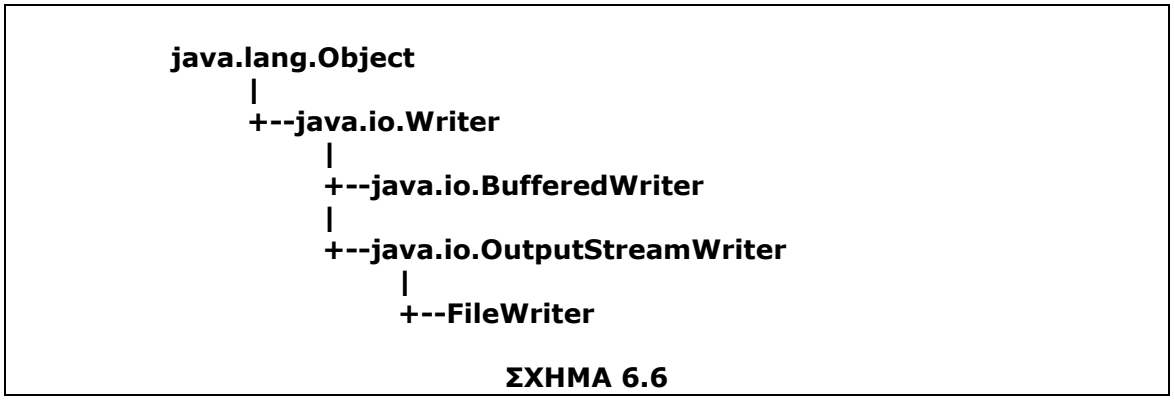
Η κλάση **FileReader** ***(public class FileReader extends InputStreamReader)*** χρησιμοποιείται για τον ορισμό των διαδικασίων ανάγνωσης αρχείων χαρακτήρων (και κειμένου).

Στο τμήμα κώδικα 5 φαίνεται η χρήση των κλάσεων και για την ανάγνωση ενός αρχείου κειμένου ανά γραμμή, μέσω της readLine().

```
import java.io.*;

public class ReadSource {
   public static void main(String[] arguments) {
      try {
         FileReader file = new FileReader("ReadSource.java");
         BufferedReader buff = new BufferedReader(file);
         boolean eof = false;
         while (!eof) {
            String line = buff.readLine();
            if (line == null)
               eof = true;
            else
               System.out.println(line);
         }
         buff.close();
      } catch (IOException e) {
         System.out.println("Error -- " + e.toString());
      }
   }
}
```

Τμήμα Κώδικα 5

Μερικές από τις βασικές κλάσεις με τις οποίες μπορεί να χειριστεί κανείς ρεύματα και αρχεία εξόδου χαρακτήρων στη Java φαίνονται παρακάτω:



ΣΧΗΜΑ 6.6

Η κλάση **Writer** ***(public abstract class Writer extends Object)*** είναι μία αφηρημένη κλάση που υλοποιεί τις διαδικασίες εξόδου από ρεύματα χαρακτήρων. Οι υποκλάσεις της κατ' αρχήν πρέπει να επαναϋλοποιήσουν τις μεθόδους **write(char[], int, int)**, **flush()**, and **close()**. Επίσης χρειάζεται να επαναυλοποιήσουν και κάποιες άλλες για λόγους αποτελεσματικότητας και λειτουργικότητας. Μερικές από τις λειτουργίες εγγραφής που ορίζονται στα πλαίασια της κλάσης **Writer** δίνονται στον πίνακα που ακολουθεί:

Βασικές Μέθοδοι της κλάσης Writer

Μέθοδος	Περιγραφή
void write(int c)	Γράφει έναν χαρακτήρα στο ρεύμα εξόδου. Ο χαρακτήρας που γράφεται περιέχεται στα 16 λιγότερο σημαντικά bits της ακέραιας τιμής c, ενώ τα 16 περισσότερο σημαντικά αγνοούνται.
void write(String str)	Γράφει το String str στο ρεύμα εξόδου.
abstract void flush( )	Εκκενώνει το ρεύμα εξόδου από τους χαρακτήρες που περιέχει. Εάν το ρεύμα έχει αποθηκεύσει κάποιους χαρακτήρες σε έναν buffer με τη βοήθεια μεθόδων write(), οι χαρακτήρες αυτοί γράφονται κατ' ευθείαν στον προόρισμό τους. Στη συνέχεια εάν η έξοδος του ρεύματος κατευθύνεται σε ένα άλλο ρεύμα και αυτό με τη σειρά του εκκενώνεται.
abstract void close()	Κλείνει το stream εξόδου αφού πρώτα το εκκενώσει.

Η κλάση **BufferedWriter** ***(public class BufferedWriter extends Writer)*** χρησιμοποιείται για την εγγραφή κειμένου σε ένα ρεύμα εξόδου χαρακτήρων με τη

μέθοδο του buffering για λόγους αποτελεσματικότητας ως προς την ταχύτητα των διαδικασιών εγγραφής. Υλοποιούνται οι διαδικασίες εγγραφής χαρακτήρα, πίνακα χαρακτήρων ή String. Η μέγεθος του buffer μπορεί να οριστεί κατά τη δημιουργία του, η εξ ορισμού τιμή του όμως είναι ικανοποιητική για τις περισσότερες προγραμματιστικές εφαρμογές.

Στα πλαίσια της **BufferedWriter** παρέχεται επίσης η μέθοδος **newLine()** η οποία δίνει τη δυνατότητα της δημιουργίας μίας αλλαγής γραμμής στο ρεύμα εξόδου χωρίς ο προγραμματιστής να είναι υποχρεωμένος να γνωρίζει τον χαρακτήρα ή τους χαρακτήρες που πρέπει να εγγραφούν για να καταχωρηθεί η αλλαγή γραμμής. Το γεγονός αυτόείναι σημαντικό για τη μεταφερσιμότητα των προγραμμάτων από σύστημα σε σύστημα. Στον πίνακα που ακολουθεί δίνονται μερικές από τις βασικές διαδικασίες εγγραφής που υλοποιούνται από την κλάση **BufferedWriter**.

Βασικές Μέθοδοι της κλάσης BufferedWriter

Μέθοδος	Περιγραφή
void write(int c)	Γράφει έναν χαρακτήρα στο ρεύμα εξόδου.
void write(String s, int i, int n)	Γράφει ένα τμήμα του String **s** στο ρεύμα εξόδου. Το τμήμα αυτό του String αρχίζει από τη θέση **i** και αποτελείται από τους **n** επόμενους χαρακτήρες.
void newLine( )	Γράφει στο ρεύμα εξόδου το ειδικό string που αντιστοιχεί σε «αλλαγή γραμμής». Το String αυτό εξαρτάται από τη μηχανή/λειτουργικό σύστημα και ορίζεται στα πλαίσια του system property σαν **line.separator**.
abstract void flush( )	Εκκενώνει το ρεύμα εξόδου από τους χαρακτήρες που περιέχει. Εάν το ρεύμα έχει αποθηκεύσει κάποιους χαρακτήρες σε έναν buffer με τη βοήθεια μεθόδων write(), οι χαρακτήρες αυτοί γράφονται κατ' ευθείαν στον προόρισμό τους. Στη συνέχεια εάν η έξοδος του ρεύματος κατευθύνεται σε ένα άλλο ρεύμα και αυτό με τη σειρά του εκκενώνεται.
abstract void close()	Κλείνει το stream εξόδου αφού πρώτα το εκκενώσει.

Ένα ρεύμα που δημιουργείται μέσω της κλάσης **OutputStreamWriter** ***(public class***

OutputStreamWriter extends Writer) χρησιμοποιείται σαν φίλτρο για την μετατροπή χαρακτήρων σε bytes τα οποία γράφονται σε ένα ρεύμα εξόδου. Η μετατροπή αυτή γίνεται με βάση κάποιο συγκεκριμένο τρόπο κωδικοποίησης. Κάθε ρεύμα τύπου **OutputStreamWriter μπορεί να ενσωματώσει τον δικό του μετατροπέα χαρακτήρων** (CharToByteConverter). Αν δεν απαιτείται ειδική κωδικοποίηση χρησιμοποιείται η εξ ορισμού (unicodes) που ορίζεται στα πλαίσια της system property σαν **file.encoding**.

Για λόγους αποτελεσματικότητας (μεγαλύτερη ταχύτητα) συνήθως ένα ρεύμα της **OutputStreamWriter** φιλτράρεται από ένα ρεύμα της, όπως για παράδειγμα:

Writer out = new BufferedWriter(new OutputStreamWriter(System.out));

Η κλάση **FileWriter** (**Public class FileWriter extends OutputStreamWriter)** χρησιμοποιείται για τον ορισμό των διαδικασιών εγγραφής αρχείων χαρακτήρων και κειμένου.

```
import java.io.*;

public class CopyFile {
   public static void main(String[] args) throws IOException {
        File inputFile = new File("farrago.txt");
        File outputFile = new File("outagain.txt");

       FileReader in = new FileReader(inputFile);
       FileWriter out = new FileWriter(outputFile);
       int charValue;

       while ((charValue = in.read()) != -1)
         out.write(charValue);

       in.close();
       out.close();
   }
}
```

Τμήμα Κώδικα 6

Στο τμήμα κώδικα 6 φαίνεται η χρήση των κλάσεων **FileReader** και **FileWriter**, για τη δημιουργία ενός αντιγράφου αρχείου κειμένου. Το αρχείο εισόδου διαβάζεται χαρακτήρα προς χαρακτήρα μέσω της **read( )** και ταυτόχρονα το αρχείομ εξόδου δημιουργείται χαρακτήρα προς χαρακτήρα μέσω της **write( )**.

Στο τμήμα κώδικα 7 φαίνεται επιπλέον η χρήση των κλάσεων **BufferedReader** και **BufferedWriter**. Το πρόγραμμα δημιουργεί κατ' αρχήν ένα νέο βοηθητικό αρχείο στο οποίο μεταφέρει το κείμενο από το αρχείο εισόδου μετατρέποντας όλους τους χαρακτήρες σε κεφαλαίους. Στη συνέχεια γράφει το βοηθητικό αρχείο στη θέση του παλιού αρχείου εισόδου.

Η χρήση των διαδικασιών που ορίζονται στα πλαίσια της κλάσης **File** στα τμήματα κώδικα 6 και 7 περιγράφεται στην επόμενη παράγραφο.

```
import java.io.*;

public class AllCapsDemo {
   public static void main(String[] arguments) {
      AllCaps cap = new AllCaps(arguments[0]);
      cap.convert();
   }
}

class AllCaps {
   String sourceName;

   AllCaps(String sourceArg) {
      sourceName = sourceArg;
   }

   void convert() {
      try {
         // Create file objects
         File source = new File(sourceName);
         File temp = new File("cap" + sourceName + ".tmp");

         // Create input stream
         FileReader fr = new FileReader(source);
         BufferedReader in = new BufferedReader(fr);
```

```
        // Create output stream
        FileWriter fw = new FileWriter(temp);
        BufferedWriter out = new BufferedWriter(fw);

        boolean eof = false;
        int inChar = 0;
        do {
           inChar = in.read();
           if (inChar != -1) {
              char outChar = Character.toUpperCase( (char)inChar );
              out.write(outChar);
           } else
              eof = true;
        } while (!eof);
        in.close();
        out.close();

        boolean deleted = source.delete();
        if (deleted)
           temp.renameTo(source);
     } catch (IOException e) {
        System.out.println("Error -- " + e.toString());
     } catch (SecurityException se) {
        System.out.println("Error -- " + se.toString());
     }
  }
}
```

Τμήμα Κώδικα 7

6.4 Διαχείριση Αρχείων μέσω της κλάσης File

Ένα στιγμιότυπο της κλάσης **File** αναπαριστά, με τη μορφή ενός string, ένα όνομα μονοπατιού (path name) που μπορεί να καθορίσει ένα συγκεκριμένο αρχείο του συστήματος αρχείων. Ορισμένες πράξεις-λειτουργίες του συστήματος αρχείων, όπως είναι η μετονομασία αρχείου και η διαγραφή αρχείου υλοποιούνται μέσω της κλάσης File και όχι μέσω των ρευμάτων (streams).

Ένα στιγμιότυπο της κλάσης **FileDescriptor** αναπαριστά, με τη μορφή μιας αφηρημένης ένδειξης, έναν προσδιοριστή αρχείου ενός συγκεκριμένου αρχείου στα πλαίσια του συστήματος αρχείων. Τέτοιου είδους προσδιοριστές αρχείων δημιουργούνται εσωτερικά από το σύστημα εισόδου/εξόδου της Java.

Δημιουργία αντικειμένων της κλάσης File

Δημιουργός	Περιγραφή
File(String pathname)	Δημιουργεί ένα νέο αρχείο μετατρέποντας τη συμβολοσειρά pathname σε ένα "abstract pathname".
File(String parent, String child)	Δημιουργεί ένα νέο αρχείο συνδυάζοντας τη συβολοσειρά **parent** που θεωρείται φάκελος και τη συβολοσειρά **child** που θεωρείται είτε φάκελος είτε αρχείο ορίζοντας το αντίστοιχο "abstract pathname"
File(File parent, String child)	Δημιουργεί ένα νέο αρχείο συνδυάζοντας το αρχείο που αντιστοιχεί στο "abstract pathname" **parent** που θεωρείται φάκελος και τη συβολοσειρά **child** που θεωρείται είτε φάκελος είτε αρχείο ορίζοντας το αντίστοιχο abstract pathname"

Βασικές Μέθοδοι της κλάσης File

Μέθοδος	Περιγραφή
boolean exists( )	Ελέγχει εάν το αρχείο που δηλώνεται από το "abstract pathname" υπάρχει.
boolean delete( )	Διαγράφει το αρχείο ή το φάκελο που ορίζεται από το "abstract pathname". Εάν πρόκειται για φάκελο τότε αυτός πρέπει να είναι κενός.
void deleteOnExit( )	Ζητά τη διαγραφή του αρχείου ή του φακελου που ορίζεται από το "abstract pathname" όταν θα τελειώσει η εκτέλεση της Java Virtual Machine.
long length( )	Επιστρέφει το μήκος του αρχείου σε bytes
boolean renameTo(File dest)	Μετονομάζει το αρχείο που ορίζεται από το "abstract pathname" στο **dest** που θα αποτελεί το νέο "abstract pathname".
boolean mkdir( )	Δημιουργει το φάκελο που ορίζεται από το "abstract pathname".
boolean mkdirs( )	Δημιουργει το φάκελο που ορίζεται από το "abstract pathname", καθώς και όλους τους φακέλους-προγόνους που απαιτούνται

6.5 Αρχεία Κατευθείαν Πρόσβασης (dirrect access files)

Ο όρος **αρχείο κατευθείαν πρόσβασης *(dirrect access file)***, χαρακτηρίζει ένα αρχείο, στο οποίο έχουμε τη δυνατότητα άμεσης πρόσβασης σ' ένα οποιδήποτε στοιχείο του. Μπορούμε δηλαδή να τοποθετήσουμε τον δείκτη του αρχείου κατ' ευθείαν στο στοιχείο που Θέλουμε να χειριστούμε. Ένα αρχείο κατευθείαν πρόσβασης ονομάζεται και **αρχείο τυχαίας πρόσβασης *(random access file)***. Την ονομασία αυτή παίρνει ένα αρχείο για να δηλωθεί έτσι η ιδιότητα που διαθέτει: «οποιοδήποτε στοιχείο του αρχείου διαλέξουμε τυχαία, χρειάζεται ίσος χρόνος για την πρόσβαση σε αυτό». Ένα επί πλέον χαρακτηριστικό των αρχείων κατευθείαν πρόσβασης είναι ότι όταν ανοίξουμε ένα αρχείο, μπορούμε να το χρησιμοποιήσουμε ταυτόχρονα σαν αρχείο εισόδου και εξόδου.

Από πλευράς χειρισμού ενός αρχείου κατευθείαν πρόσβασης, κατ' αρχήν δεν υπάρχει καμμία διαφοροποίηση σε σχέση με τα ακολουθιακά αρχεία. Αυτό σημαίνει ότι μπορούμε να χειριστούμε κατ΄ αρχήν ένα τέτοιο αρχείο ακολουθιακά. Επιπλέον όμως για να είναι δυνατός ο χειρισμός του και με διαδικασίες κατευθείαν πρόσβασης πρέπει να ισχύουν κατ' ελάχιστο τα εξής:

- Να είναι γνωστός ο τρόπος αρίθμησης των στοιχείων του αρχείου. Συνήθως η σχετική Θέση του πρώτου στοιχείου είναι 0 (μηδέν), ενώ η σχετική Θέση του τελευταίου στοιχείου ενός αρχείου που περιέχει Ν στοιχεία είναι Ν-1.
- Να υπάρχει η δυνατότητα τοποθέτησης του δείκτη του αρχείου κατευθείαν σε οποιοδήποτε στοιχείο.
- Να υπάρχει η δυνατότητα να μάθουμε σε ποιο στοιχείο του αρχείου είναι τοποθετημένος ο δείκτης του αρχείου.

Στη Java η κλάση **RandomAccessFile** παρέχει τις απαραίτητες μεθόδους για την επεξεργασία αρχείων κατ' ευθείαν πρόσβασης. Ένα αρχείο κατευθείαν πρόσβασης «συμπεριφέρεται» σαν ένας μεγάλος πίνακας από bytes ο οποίος είναι αποθηκευμένος στη δευτερεύουσα μνήμη. Για κάθε αρχείο τον αντίστοιχο ρόλο του ενδείκτη του πίνακα παίζει ο δείκτης του αρχείου (file-pointer), ο οποίος καθορίζει το σημείο του αρχείου στο οποίο θα γίνει η επόμενη διαδικασία εισόδου ή

εξόδου. Ο δείκτης του αρχείου γνωρίζει την τρέχουσα θέση σαν ένα αριθμό από bytes που δηλώνει την απόσταση από την αρχή (offset) του αρχείου (byte 0).

Οι μέθοδοι (constructors) για τη δημιουργία ενός ρεύματος που αντιστοιχεί σε ένα αρχείο κατευθείαν πρόσβασης είναι οι εξής:

Μέθοδοι (constructors) για τη δημιουργία ενός αρχείου RandomAccessFile

Μέθοδος	Περιγραφή
RandomAccessFile(String file, String mode)	Δημιουργεί ένα ρεύμα που αντιστοιχεί στο αρχείο κατ' ευθείαν πρόσβασης με όνομα **file**. Το ρεύμα αυτό μπορεί να είναι είτε ρεύμα εισόδου είτε ρεύμα εισόδου και εξόδου ταυτόχρονα, ανάλογα με την τιμή της παραμέτρου **mode**. Η **mode** μπορεί να έχει την τιμή **"r"** οπότε πρόκειται για ρεύμα εισόδου ή **"rw"** οπότε πρόκειται για ρεύμα εισόδου/εξόδου.
RandomAccessFile(File file, String mode)	Όπως παραπάνω με τη διαφορά ότι το αντικείμενο **file** έχει οριστεί προηγουμένως μέσω της κλάσης **File**

Έτσι για παράδειγμα η:

```
RandomAccessFile RAFile1 = new RandomAccessFile("myfile1.dat", "r");
```

δημιουργεί το ρεύμα **RAFile1** που αντιστοιχεί στο αρχείο κατευθείαν πρόσβασης **myfile1.dat**, στο οποίο μπορούν να εφαρμοστούν μόνο διαδικασίες εισόδου, ενώ η:

```
RandomAccessFile RAFile2 = new RandomAccessFile("myfile2.dat", "rw");
```

δημιουργεί το ρεύμα **RAFile2** που αντιστοιχεί στο αρχείο κατευθείαν πρόσβασης **myfile2.dat**, στο οποίο μπορούν να εφαρμοστούν και διαδικασίες εισόδου αλλά και διαδικασίες εξόδου.

Η δυνατότητα ανοίγματος ενός αρχείου ταυτόχρονα και για είσοδο και για έξοδο δεδομένων διαφοροποιεί τα αρχεία κατευθείαν πρόσβσης από τα ακολουθιακά αρχεία, τα οποία είναι κατ' αποκλειστικότητα είτε μόνον εισόδου είτε μόνον εξόδου.

Σε σχέση με το μηχανισμό υλοποίησης των διαδικασιών κατευθείαν πρόσβασης που διαθέτει η Java, η τιμή του δείκτη του αρχείου μπορεί να διαβαστεί με τη βοήθεια της μεθόδου **getFilePointer** και μπορεί να τοποθετηθεί με τη βοήθεια της μεθόδου **seek**.

Βασικές Μέθοδοι της Κλάσης RandomAccessFile
που αναφέρονται στην κατευθείαν πρόσβαση

Μέθοδος	Περιγραφή
void seek(long pos)	Τοποθετεί τον δείκτη του αρχείου ακριβώς πριν από το byte που αναφέρεται στη θέση **pos**
long getFilePointer( )	Επιστρέφει τη θέση που δείχνει ο δείκτης του αρχείου
int skipBytes(int number)	Μετακινεί το δείκτη του αρχείου κατά τον αριθμό των bytes που αναφέρεται στη παράμετρο **number**

Οι μέθοδοι εισόδου διαβάζουν bytes από ένα αρχείο κατευθείαν πρόσβασης, ξεκινώντας από την τρέχουσα θέση του δείκτη του αρχείου, μετακινώντας τον κατά τον αριθμό των bytes που διαβάστηκαν.

Οι μέθοδοι εξόδου γράφουν bytes σε ένα αρχείο κατευθείαν πρόσβασης, ξεκινώντας από την τρέχουσα θέση του δείκτη του αρχείου, μετακινώντας τον στην αμέσως επόμενη θέση μετά το τελευταίο byte που γράφηκε.

Εάν μία μέθοδος εξαναγκαστεί να γράψει πέραν του τέλους του αρχείου, προκαλεί την απαραίτητη αλλαγή στην τιμή του δείκτη του αρχείου και προκαλεί επέκταση του αρχείου κατά τον ανάλογο αριθμό από bytes.

Γενικά η κλάση RandomAccesFile διαθέτει μεθόδους για την ανάγνωση και εγγραφή όλων των πρωταρχικών τύπων δεδομένων, καθώς και δεδομένων τύπου String, όπως φαίνεται στους παρακάτω πίνακες:

Μέθοδοι που υλοποιούν διαδικασίες εισόδου της Κλάσης RandomAccessFile

Μέθοδος	Περιγραφή
long length( )	Επιστρέφει το μήκος του αρχείου σαν ένα αριθμό από bytes
int read( )	Διαβάζει ένα byte από το αρχείο, το οποίο επιστρέφεται σαν ένας ακέραιος στο διάστημα 0-255
int read(byte[ ] b)	Διαβάζει δεδομένα μέχρι **b.length** bytes από το αρχείο και τα τοποθετεί στον πίνακα **b**. Επιστρέφει τον αριθμό των bytes που διάβασε ή –1 εάν συναντήσει τέλος αρχείου.
boolean readBoolean( )	Διαβάζει και επιστρέφει ένα δεδομένο τύπου boolean από το αρχείο.
byte readByte( )	Διαβάζει ένα byte από το αρχείο σαν μία προσημασμένη τιμή των 8 bits
char readChar( )	Διαβάζει τα επόμενα 2 bytes από το αρχείο και τα επιστρέφει σαν έναν Unicode χαρακτήρα
int readInt( )	Διαβάζει τα επόμενα 4 bytes από το αρχείο, τα οποία επιστρέφει σαν έναν ακέραιο προσημασμένο αριθμό των 32 bits.
float readFloat( )	Διαβάζει τα επόμενα 4 bytes από το αρχείο, τα οποία επιστρέφει σαν έναν πραγματικό αριθμό τύπου **float**.
double readDouble( )	Διαβάζει τα επόμενα 8 bytes από το αρχείο, τα οποία επιστρέφει σαν έναν πραγματικό αριθμό τύπου **double**.
String readLine( )	Διαβάζει διαδοχικά τα επόμενα bytes από το αρχείο ξεκινώντας από αυτό που δείχνει ο δείκτης του αρχείου, μέχρι να συναντήσει τέλος γραμμής ή τέλος αρχείου. Η μέθοδος αυτή δεν υποστηρίζει το πλήρες σύνολο των Unicode χαρακτήρων.

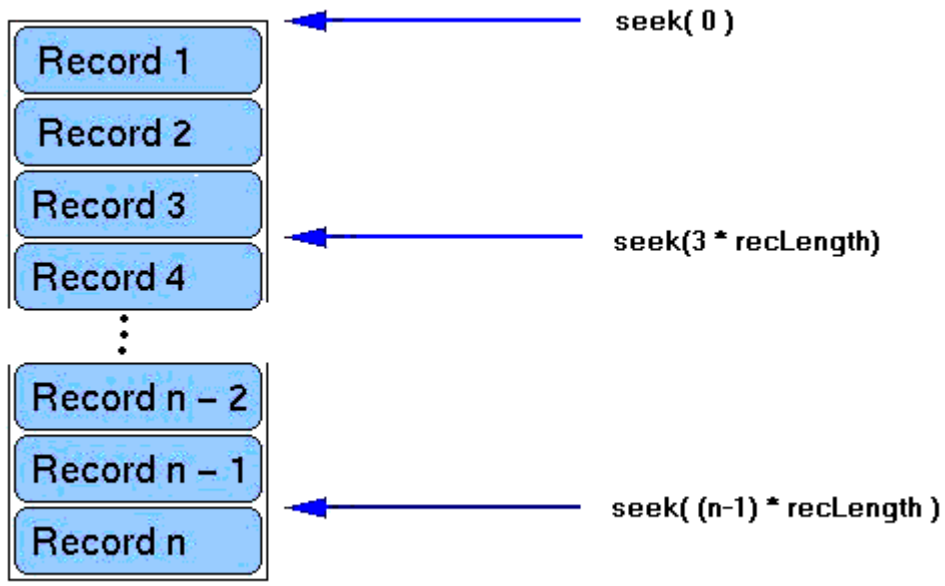
Μέθοδοι που υλοποιούν διαδικασίες εξόδου της Κλάσης RandomAccessFile

Μέθοδος	Περιγραφή
void setLength(long newLength)	Τοποθετεί το μήκος του αρχείου με **newLength**. Εάν το τρέχον μήκος του αρχείου είναι μεγαλύτερο από **newLength** το μέρος του αρχείου που απομένει αποκόπτεται. Εάν είναι μικρότερο από **newLength** τότε το αρχείο επεκτείνεται. Στη δεύτερη περίπτωση οι τιμές των bytes που προστίθενται είναι μη-ορισμένες.
void write(int b)	Γράφει το byte **b** στο αρχείο
void write(byte[ ] b)	Γράφει **b.length** bytes από τον πίνακα **b** στο αρχείο αρχίζοντας από την τρέχουσα θέση του δείκτη του αρχείου.
void writeChar(int ch)	Γράφει το χαρακτήρα **ch** σαν μία τιμή των 2 bytes, με το πιο σημαντικό byte να γράφεται πρώτο.
void writeBoolean(boolean b)	Γράφει την boolean τιμή **b** σαν μία τιμή του ενός byte στο αρχείο.
void writeInt(int v)	Γράφει τον ακέραιο **v** σαν μία τιμή των 4 bytes στο αρχείο.
void writeFloat(float v)	Μετατρέπει τον πραγματικό αριθμό τύπου float σε έναν ακέραιο (int) χρησιμοποιώντας τη μέθοδο **floatToIntBits** της κλάσης **Float**, και την γράφει σαν έναν ακέραιο των 4 bytes (int) στο αρχείο, ξεκινώντας από το πιο σημαντικό byte.
void writeDouble(double v)	Μετατρέπει τον πραγματικό αριθμό τύπου **double** σε έναν ακέραιο (int) χρησιμοποιώντας τη μέθοδο **doubleToLongBits** της κλάσης **Double**, και τον γράφει σαν έναν ακέραιο των 8 bytes (long) στο αρχείο, ξεκινώντας από το πιο σημαντικό byte.
void writeBytes(String s)	Γράφει το string s σαν μία ακολουθία από bytes στο αρχείο. Κάθε χαρακτήρας γράφεται στο αρχείο αφού του αποκοπεί το πρώτο byte.

Η υλοποίηση ενός αρχείου κατ' ευθείαν πρόσβασης στις περισσότερες περιπτώσεις προϋποθέτει την ύπαρξη εγγραφών σταθερού μήκους, καθώς με αυτόν τον τρόπο γίνεται απλός ο υπολογισμός της θέσης της κάθε εγγραφής.

Στις διάφορες προγραμματιστικές εφαρμογές, ένα αρχείο κατ' ευθείαν πρόσβασης δεν διαφέρει, σε φυσικό επίπεδο υλοποίησης, από ένα αρχείο, του οποίου τα στοιχεία διαβάζονται ή γράφονται ακολουθιακά. Η διαφορά έγγειται στον τρόπο οργάνωσής του, στη μέθοδο δηλαδή, με την οποία τοποθετούνται τα στοιχεία στο αρχείο. Η τοποθέτηση πρέπει να γίνει με τέτοιο τρόπο, ώστε να υπάρχει κάποια στενή σχέση ανάμεσα στην τιμή του στοιχείου και στη σχετική του Θέση στο αρχείο.
Ας υποθέσουμε (κάτι που ισχύει στις περισσότερες εφορμογές), ότι τα στοιχεία του αρχείου είναι τύπου εγγραφής *(record)*, υλοποιούνται δηλαδή στα πλαίσια της Java σαν ένα αντικείμενο (τύπος κλάσης).

Η υλοποίηση ενός αρχείου κατ' ευθείαν πρόσβασης στις περισσότερες περιπτώσεις προϋποθέτει την ύπαρξη εγγραφών σταθερού μήκους, καθώς με αυτόν τον τρόπο γίνεται απλός ο υπολογισμός της θέσης της κάθε εγγραφής. Στο σχήμα 6.6 φαίνεται ο υπολογισμός της θέσης της 1ης, 4ης και n-οστής εγγραφής ενός αρχείου από bytes.



Σχ. 6.6: Υπολογισμός θέσης μίας εγγραφής

Στην περίπτωση αυτή η πιό βασική μας δουλειά, πρίν να αποφασίσουμε τη σειρά τοποθέτησης των στοιχείων στο αρχείο, είναι να ορίσουμε μία συνάρτηση που να συνδέει την τιμή ενός συγγεκριμένου πεδίου κάθε εγγραφής με τη σχετική Θέση αυτής της εγγραφής στο αρχείο. Το πεδίο αυτό με βάση το οποίο μπορούμε να υπολογίσουμε τη θέση της εγγραφής στο αρχείο ονομάζεται κλειδί *(key),* και η συνάρτηση υπολογισμού της θέσης ονομάζεται **συνάρτηση "hash".**

Από τη στιγμή που γνωρίζουμε πώς μπορούμε να υπολογίσουμε τη Θέση μιάς εγγραφής με βάση το πεδίο-κλειδί, μπορούμε να δημιουργήσουμε το αρχείο, όπως ακριβώς Θα δημιουργούσαμε ένα οποιοδήποτε ακολουθιακό αρχείο. Μια κατάλληλη τεχνική συνίσταται στα εξής βήματα:

(α) Ανοίγουμε το αρχείο, σαν αρχείο εξόδου.

(β) Γράφουμε στο αρχείο ένα προκαθορισμένο αριθμό από εγγραφές, οι οποίες περιέχουν ουσιαστικά τιμή, μόνο στο πεδίο κλειδί. Ολα τα άλλα πεδία έχουν αρχικά κάποιες εικονικές τιμές, όπως γιά παράδειγμα μηδέν ή κενό.

(γ) Κλείνουμε το αρχείο.

Με τον τρόπο αυτό το αρχείο (ας το ονομάσουμε **F**) είναι "έτοιμο" να δεχτεί, στη συνέχεια, τις πραγματικές τιμές των εγγραφών με οποιαδήποτε σειρά Θελήσουμε και όταν αυτές γίνουν διαθέσιμες.

Για να τοποθετήσουμε τις πραγρατικές τιμές σε μιά εγγραφή του αρχείου ακολουθούμε το εξής "σενάριο":

(α) Ανοίγουμε το αρχείο **F** σαν ρεύμα εισόδου και εξόδου.

(β) Με βάση την τιμή του πεδίου-κλειδιού, υπολογίζουμε τη σχετική θέση **N,** της εγγραφής στο αρχείο και τοποθετούμε το δείκτη του αρχείου στη Θέση αυτή με τη βοήθεια της **seek(N)**

(γ) Διαβάζουμε την εγγραφή από το αρχείο.

(δ) Τοποθετούμε τις τιμές που Θέλουμε στα αντίστοιχα πεδία, εκτός φυσικά από το πεδίο-κλειδί, η τιμή του οποίου παραμένει η ίδια.

(ε) Επαναφέρουμε το δείκτη του αρχείου **F** στη Θέση **N**, με τη βοήθεια της **seek(N)**. Η ενέργεια αυτή είναι απαραίτητη γιατί ο δείκτης έχει τοποθετηθεί στη Θέση **N+recLength**, όπου **recLength** το μήκος της εγγραφής σε bytes) εξ' αιτίας της διαδικασίας ανάγνωσης.

(ζ) Ξαναγράφουμε την εγγραφή στο αρχείο **F**.

Εφαρμογή:

Ενα φαρμακείο Θέλει να κρατάει σε ένα αρχείο (κατ' ευθείαν πρόσβασης) πληροφορίες για τα φάρμακα που έχει στα ράφια του. Κάθε φάρμακο χαρακτηρίζεται από έναν αύξοντα αριθμό (έστω από το 0 μέχρι και το 99), από το όνομά του, από τον τύπο του και από την τιμή του. Οι βασικές λειτουργίες που ορίζονται για το αρχείο των φαρμάκων είναι οι εξής:

[1] Τοποθέτηση ενός φαρμάκου στο αρχείο

[2] Διαγραφή ενός φαρμάκου από το αρχείο

[3] Παρουσίαση των πληροφοριών ενός φαρμάκου

[4] Αλλαγή της τιμής ενός φαρμάκου

Ολες οι παραπάνω λειτουργίες γίνονται με βάση τον αύξοντα αριθμό του φαρμάκου (πεδίο-κλειδί). Θεωρούμε δηλαδή ότι η συνάρτηση hash για τον υπολογισμό της θέσης της εγγραφής πάνω στο αρχείο στην περίπτωση αυτή είναι απλά hash(X) = X.

Στο τμήμα κώδικα 8 δίνεται η υλοποίηση του τύπου φάρμακο με τη βοήθεια της κλάσης Medicine, και στο τμήμα κώδικα 9 δίνεται η υλοποίηση της κλάσης RAMedicine η οποία επεκτείνει την Medicine, έτσι ώστε να υλοποιείται μία διαδικασία για την ανάγνωση ενός φαρμάκου από ένα αρχείο κατευθείαν πρόσβασης και μία διαδικασία για την εγγραφή ενός φαρμάκου στο αρχείο.

```
public class Medicine
{
      private int mediCode;
      private String mediName;
      private String mediType
      private double mediPrice;

        public Medicine()
        {
            this(0, "", "", 0.0);
        }

        public Medicine(int code, String name, String type, double price)
        {
            setMediCode(code);
            setMediName(name);
            setMediType(type);
```

```
            setMediprice(price);
        }

        public void setMediCode (int code)
        {
            mediCode = code;
        }

        public int getMediCode( )
        {
            return mediCode
        }

        public void setMediName (int name)
        {
            mediName = name;
        }

        public int getMediName( )
        {
            return mediName
        }

        public void setMediType (int type)
        {
            mediType = type;
        }

        public int getMediType( )
        {
            return mediType
        }

        public void setMediPrice (int price)
        {
            mediPrice = price;
        }

        public int getMediPrice( )
        {
            return mediPrice
        }
}
```

Τμήμα Κώδικα 8

```
public class RAMedicine extends Medicine
{
        public RAMedicine()
        {
            this(0, "", "", 0.0);
        }

        public RAMedicine(int code, String name, String type, double price)
        {
            super(code, name, type, price);
        }

        public RAMedicine read(RandomAccessFile file) throws IOException
        {
            setMediCode(file.readInt());
            setMediName(cutName(file));
            setMediType(cutName(file));
            setMediprice(file.readDouble());
            return this;
        }
        private String cutName(RandomAccessFile file) throws IOException
        {
            char stringName[ ] = new char[10];
            for ( int i = 0; I < stringName.length, i++ )
            { stringName[i] = file.readChar( ); }
            return new String(stringName).replace('\0', ' ');
        }

        public void write(RandomAccessFile file) throws IOException
        {
            file.writeInt(getMediCode());
            putName(file, getMediName());
            putName(file, getMediType());
            file.writeDouble(getMediPrice());
        }

        private void putName(RandomAccessFile file String stringName)
                                                              throws IOException
        {
            StringBuffer buf =null;
            if (stringName != null)
               buf = new StringBuffer(stringName);
            else
               buf = new StringBuffer( 10 );
            buf.setLength( 10 );
            file.writeChars( buftoString());
        }

        public static int size( ) {
            return 52;
        }
}
```

Τμήμα Κώδικα 9

ΠΑΡΑΡΤΗΜΑ 1:

Η Ιεραρχία των Κλάσεων και Διασυνδέσεων του πακέτου java.io

Class Hierarchy

- class java.lang.**Object**
  - class java.io.**File** (implements java.lang.Comparable, java.io.Serializable)
  - class java.io.**FileDescriptor**
  - class java.io.**InputStream**
    - class java.io.**ByteArrayInputStream**
    - class java.io.**FileInputStream**
    - class java.io.**FilterInputStream**
      - class java.io.**BufferedInputStream**
      - class java.io.**DataInputStream** (implements java.io.DataInput)
      - class java.io.**LineNumberInputStream**
      - class java.io.**PushbackInputStream**
    - class java.io.**ObjectInputStream** (implements java.io.ObjectInput, java.io.ObjectStreamConstants)
    - class java.io.**PipedInputStream**
    - class java.io.**SequenceInputStream**
    - class java.io.**StringBufferInputStream**
  - class java.io.**ObjectInputStream.GetField**
  - class java.io.**ObjectOutputStream.PutField**
  - class java.io.**ObjectStreamClass** (implements java.io.Serializable)
  - class java.io.**ObjectStreamField** (implements java.lang.Comparable)
  - class java.io.**OutputStream**
    - class java.io.**ByteArrayOutputStream**
    - class java.io.**FileOutputStream**
    - class java.io.**FilterOutputStream**
      - class java.io.**BufferedOutputStream**
      - class java.io.**DataOutputStream** (implements java.io.DataOutput)
      - class java.io.**PrintStream**
    - class java.io.**ObjectOutputStream** (implements java.io.ObjectOutput, java.io.ObjectStreamConstants)
    - class java.io.**PipedOutputStream**

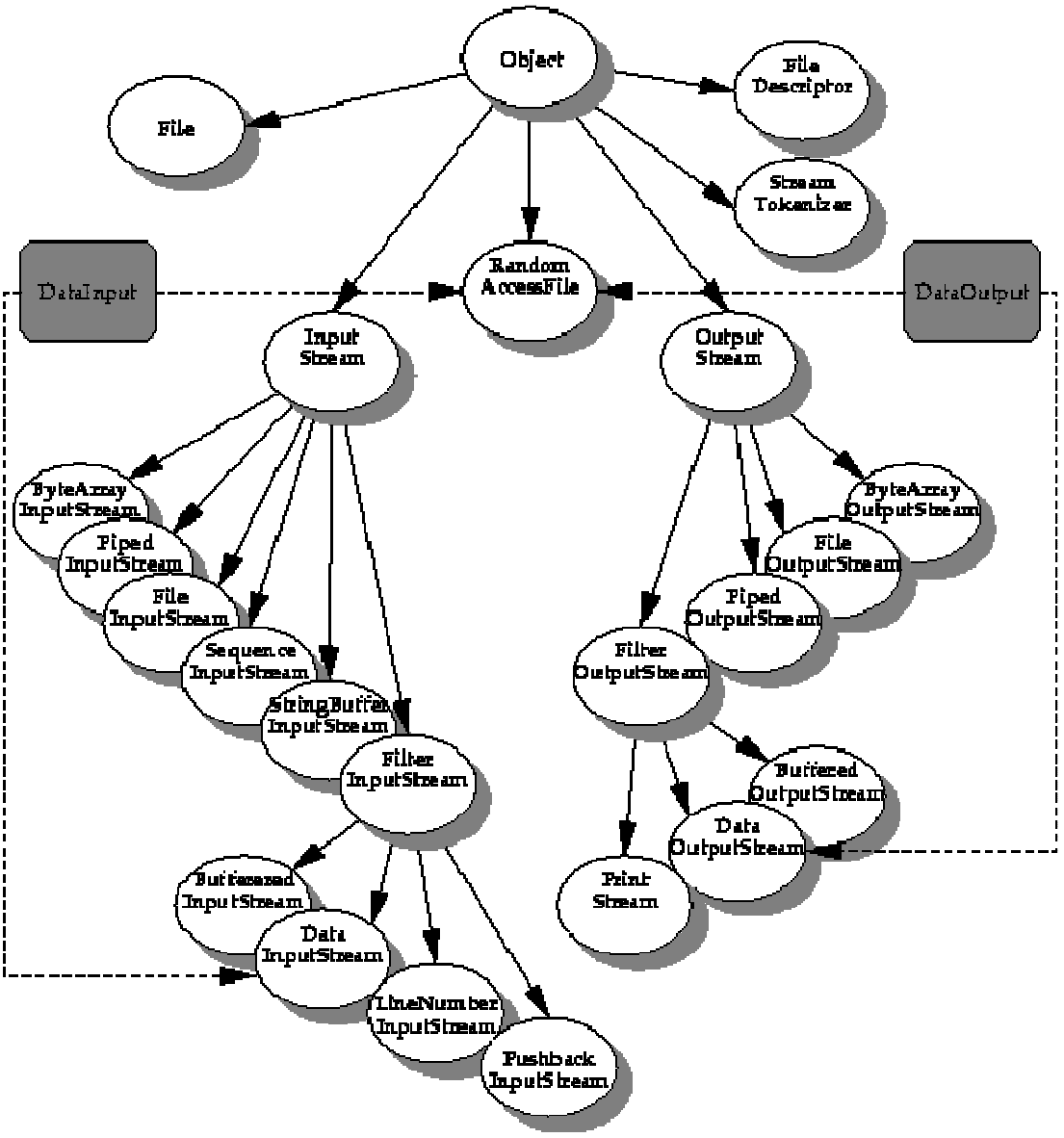
- class java.**security.Permission** (implements java.security.Guard, java.io.Serializable)
  - class java.security.**BasicPermission** (implements java.io.Serializable)
    - class java.io.**SerializablePermission**
  - class java.io.**FilePermission** (implements java.io.Serializable)
- class java.io.**RandomAccessFile** (implements java.io.DataInput, java.io.DataOutput)
- class java.io.**Reader**
  - class java.io.**BufferedReader**
    - class java.io.**LineNumberReader**
  - class java.io.**CharArrayReader**
  - class java.io.**FilterReader**
    - class java.io.**PushbackReader**
  - class java.io.**InputStreamReader**
    - class java.io.**FileReader**
  - class java.io.**PipedReader**
  - class java.io.**StringReader**
- class java.io.**StreamTokenizer**
- class java.**lang.Throwable** (implements java.io.Serializable)
  - class java.**lang.Exception**
    - class java.io.**IOException**
      - class java.io.**CharConversionException**
      - class java.io.**EOFException**
      - class java.io.**FileNotFoundException**
      - class java.io.**InterruptedIOException**
      - class java.io.**ObjectStreamException**
        - class java.io.**InvalidClassException**
        - class java.io.**InvalidObjectException**
        - class java.io.**NotActiveException**
        - class java.io.**NotSerializableException**
        - class java.io.**OptionalDataException**
        - class java.io.**StreamCorruptedException**
        - class java.io.**WriteAbortedException**
      - class java.io.**SyncFailedException**
      - class java.io.**UnsupportedEncodingException**
      - class java.io.**UTFDataFormatException**

- class java.io.**Writer**
  - class java.io.**BufferedWriter**
  - class java.io.**CharArrayWriter**
  - class java.io.**FilterWriter**
  - class java.io.**OutputStreamWriter**
    - class java.io.**FileWriter**
  - class java.io.**PipedWriter**
  - class java.io.**PrintWriter**
  - class java.io.**StringWriter**

Interface Hierarchy

- interface java.io.**DataInput**
  - interface java.io.**ObjectInput**
- interface java.io.**DataOutput**
  - interface java.io.**ObjectOutput**
- interface java.io.**FileFilter**
- interface java.io.**FilenameFilter**
- interface java.io.**ObjectInputValidation**
- interface java.io.**ObjectStreamConstants**
- interface java.io.**Serializable**
  - interface java.io.**Externalizable**

Βλέπε
https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html
για την πλήρη περιγραφή των κλάσεων που υλοποιούν είσοδο/έξοδο


Object
File
File Descriptor
Stream Tokenizer
DataInput
Random AccessFile
DataOutput
Input Stream
Output Stream
ByteArray InputStream
Piped InputStream
File InputStream
Sequence InputStream
StringBuffer InputStream
Filter InputStream
Buffered InputStream
Data InputStream
LineNumber InputStream
Pushback InputStream
ByteArray OutputStream
File OutputStream
Piped OutputStream
Filter OutputStream
Buffered OutputStream
Data OutputStream
Print Stream