ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS
ΚΕΝΤΡΟ ΕΠΙΜΟΡΦΩΣΗΣ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗΣ

Είσοδος/Έξοδος δεδομένων

Αθ. Ανδρούτσος

Συστήματα αρχείων

- Το σύστημα αρχείων όλων των Λειτουργικών συστημάτων είναι ιεραρχικό με δενδρική μορφή
- Βασικές δομές είναι οι ***φάκελοι (folders) και τα αρχεία (files)***
- Οι φάκελοι των Windows στην ορολογία των UNIX-Like, Linux και macOS συστημάτων ονομάζονται ***ευρετήρια ή κατάλογοι (directories)***

Φάκελοι και ευρετήρια

- Οπότε οι όροι **φάκελος, ευρετήριο, κατάλογος** μπορούν να χρησιμοποιούνται εναλλακτικά εννοώντας χώρους ομαδοποίησης αρχείων ή άλλων υποκαταλόγων
- Στην ιεραρχία ενός συστήματος αρχείων ο υψηλότερος κατάλογος ονομάζεται ***ρίζα***
- Στα Windows ο υψηλότερος στην ιεραρχία φάκελος (root folder) είναι ο ***C:***
- Στα UNIX-Like, Linux και Mac συστήματα το root directory είναι το ***/***

Γνωστοί κατάλογοι

- Στα Windows το **home folder ενός χρήστη** είναι το **C:\Users*username*** ενώ στα Linux-based συστήματα το home dir είναι **/home/*username*** και **/Users/username** σε macOS
- Τα παραπάνω user home directories έχουν ψευδώνυμο (alias) ~
- Μπορούμε να δημιουργήσουμε και τους δικούς μας καταλόγους, π.χ. C:\Users\a8ana\aueb στα Windows ή /home/a8ana/aueb στα Linux-based

Αρχεία και διαδρομές αρχείων

- Το **πλήρες όνομα ενός αρχείου ή φακέλου** από τη ρίζα ονομάζεται μονοπάτι (***path***)
- Για παράδειγμα το ***C:\Users\a8ana\docs*** είναι ένα μονοπάτι (path). Επίσης το path ενός αρχείου είναι το: ***C:\Users\thanos\docs\tmp.doc***
- Στα Linux ***/home/a8ana/docs/tmp.doc***
- *Σε macOS* ***/Users/a8ana/docs/tmp.doc***
- Η Java επιτρέπει να αναφερόμαστε σε μονοπάτια με **διαχωριστικό το /** ακόμα και για Windows-based συστήματα

Java I/O

- Το I/O σημαίνει **Input / Output**. Το Java I/O μας δίνει τη δυνατότητα να επικοινωνήσουμε με αρχεία (αλλά και δικτυακές ροές)
- Οι δύο βασικές πράξεις που μπορούμε να πραγματοποιήσουμε με αρχεία είναι:
  - **Read.** Διάβασμα περιεχομένων αρχείων κειμένου ή binary data (π.χ. αρχεία εικόνας)
  - **Write.** Γράψιμο (δημιουργία ή προσθήκη) σε αρχεία κειμένου ή binary data

Java I/O, Java NIO

- Το Java I/O δημιουργήθηκε στις αρχικές εκδόσεις της Java ήδη από το 1995 και μετά
- Το 2002, στην Java 1.4 αναπτύχθηκε το NIO (New I/O) που δίνει τη δυνατότητα ασύγχρονης επικοινωνίας με αρχεία και δικτυακές ροές
- Το 2011, στην Java 1.7, εκδόθηκε το NIO2 με περισσότερες δυνατότητες

I/O

- Για την είσοδο/έξοδο δεδομένων από και προς διάφορα αρχεία η Java παρέχει τα:
- ***java.io*** *package* (Java 1.0) και
- ***java.nio.file*** package (Java 1.4 και Java 1.7 updated)
- Το βασικό improvement του nio (new IO) είναι ότι το **nio είναι async** δηλαδή μπορεί ταυτόχρονα ένα thread να κατεβάζει κάτι από το δίκτυο και μέχρι να κατέβει να μην ΄περιμένει' όπως κάνει το java.io αλλά ταυτόχρονα να συνεχίζει να εκτελείται (non-blocking mode)

java.io

- Η βασική κλάση στο java.io είναι η κλάση **File** για να **αναπαριστούμε αρχεία ή directories:**

```
File fd = new File("C:/tmp/file6.txt");
```

- Παραπάνω δηλώνουμε μία μεταβλητή αρχείου **fd** (file descriptor) και συσχετίζουμε με το εξωτερικό αρχείο C:/tmp/file6.txt. Στην Java μπορούμε να δηλώνουμε file paths με / ανεξάρτητα από το λειτουργικό σύστημα. Σε Windows το C:/tmp/file6/txt θα μεταφραστεί σε C:\tmp\file6.txt
- Για να ελέγξουμε αν μία μεταβλητή τύπου File είναι αρχείο ή directory παρέχονται οι isFile(), isDirectrory()
- Γενικά με την java.io μπορούμε να δημιουργήσουμε αρχεία και καταλόγους, να ελέγξουμε αν υπάρχουν, να διαγράψουμε, να μετονομάσουμε, να λάβουμε το όνομα του αρχείου ή του directory ή του parent directory

java.io

- Στο java.io μπορούμε να γράψουμε text αρχεία με ***FileWriter, BufferedWriter, PrintStream, PrintWriter*** **και** ή να διαβάσουμε text αρχεία με ***Scanner, FileReader*** **και** ***BufferedReader***
- Binary αρχεία μπορούμε να γράφουμε με ***FileOutputStream, BufferedOutputStream*** και να διαβάσουμε με ***FileInputStream*** και ***BuffredInputStream***

Βασικοί Τύποι Ροών

- Οι βασικές ροές αφορούν το διάβασμα ή γράψιμο **χαρακτήρων** ή **row bytes** (εικόνες, βίντεο, PDF, κλπ.)

	READING	WRITING
Ροές Χαρακτήρων (Streams of Character)	**Reader**	**Writer**
Ροές Bytes (Streams of Bytes)	**InputStream**	**OutputStream**

- Οι βασικές κλάσεις της Java για I/O είναι κλάσεις των παραπάνω τύπων

Αντιστοίχιση στην Java

Βασικές Δομές I/O	Αντίστοιχες Δομές στην Java
Ροές	***InputStream*** – Γενική Ροή Εισόδου ***OutputStream*** – Γενική Ροή Εξόδου
Αρχεία row bytes	***FileInputStream*** -- Χρησιμοποιείται για είσοδο αμορφοποίητων δεδομένων **BufferedInputStream** -- Χρησιμοποιεί εσωτερικό buffer ***FileOutputStream*** -- Χρησιμοποιείται για έξοδο **BufferedOutputStream** -- Χρησιμοποιεί εσωτερικό buffer
Αρχεία Κειμένου	***FileReader (char-by-char) / BufferedReader (Large text files)*** ***Scanner (Primitives, Structured Data) /*** -- Χρησιμοποιούνται για είσοδο δεδομένων χαρακτήρων ***FileWriter (char-by-char) / BufferedWriter / PrintWriter / PrintStream*** -- Χρησιμοποιούνται για έξοδο δεδομένων χαρακτήρων

Standard Input, Output, Error

- Το πληκτρολόγιο θεωρείται ως η standard ροή εισόδου (***System.in*** στην Java) ενώ η οθόνη ως η standard ροή εξόδου (***System.out*** στην Java)
- Επίσης, υπάρχει και μία ακόμα standard ροή, η ροή εξόδου λαθών (***System.err*** στην Java), που πάλι αντιστοιχίζεται στην οθόνη και χρησιμοποιείται όταν θέλουμε να εμφανίσουμε μηνύματα λάθους ή Log άμεσα

FileWriter (1)


```java
package gr.aueb.cf.ch9;

import ...

public class File3FileWriterPrintStream8 {

    public static void main(String[] args) {
        File fd = new File("C:/tmp/file-writer.txt");
        try {
            fileWriter(fd);
        } catch (IOException e) {
            System.out.println("Το αρχείο δεν δημιουργήθηκε. ");
        }

    }

    public static void fileWriter(File file) throws IOException {
        try (FileWriter fw = new FileWriter(file)) {
            fw.write("Coding Factory\n");
            fw.flush();
        } catch (IOException e) {
            System.out.println(LocalDateTime.now() + "\n" + e);
            throw e;
        }
    }
```

- Ο FileWriter είναι **πολύ αργός γιατί γράφει char-by-char**
- Στην συγκεκριμένη μορφή δημιουργεί κάθε φορά που τρέχει, ένα νέο αρχείο
- Με .flush() διασφαλίζουμε ότι γράφει άμεσα στο output χωρίς ενδιάμεσο buffering που δημιουργεί καθυστερήσει στην έξοδο του output

FileWriter (2)


```java
public static void fileWriter(File file) throws IOException {
    try (FileWriter fw = new FileWriter(file, true)) {
        fw.write("Coding Factory\n");
        fw.flush();
    } catch (IOException e) {
        System.out.println(LocalDateTime.now() + "\n" + e);
        throw e;
    }
}
```

- Με 2η παράμετρο true ενεργοποιείται το append. Δεν δημιουργείται νέο αρχείο κάθε φορά που τρέχει, αλλά τα data προστίθενται στο τέλος του υπάρχοντος αρχείου

BufferedWriter


```java
public static void main(String[] args) {
    File fd = new File("C:/tmp/file-writer.txt");
    try {
        bufferedWriter(fd);
    } catch (IOException e) {
        System.out.println("Το αρχείο δεν δημιουργήθηκε. ");
    }

}

public static void bufferedWriter(File file) throws IOException {
    try (BufferedWriter bw = new BufferedWriter(new FileWriter(file))) {
        bw.write("Coding!");
        bw.newLine();
        bw.flush();
    } catch (IOException e) {
        System.out.println(LocalDateTime.now() + "\n" + e);
        throw e;
    }
}
```

- Ο BufferedWriter **είναι efficient** γιατί χρησιμοποιεί ένα εσωτερικό buffer **8196 bytes**
- Επομένως με ένα write γράφει 8196 characters
- Είναι wrapper του *FileWriter*, όπως φαίνεται από το new

PrintStream


```
public class File3FileWriterPrintStream8 {

    public static void main(String[] args) {
        File fd = new File("C:/tmp/file-writer.txt");
        try {
            bufferedWriter(fd);
        } catch (IOException e) {
            System.out.println("Το αρχείο δεν δημιουργήθηκε. ");
        }
    }

    public static void printStream(String file) throws IOException {
        try (PrintStream ps = new PrintStream(file)) {
            ps.println("Using PrintStream");
            ps.flush();
        } catch (IOException e) {
            System.out.println(LocalDateTime.now() + "\n" + e);
            throw e;
        }
    }
```

- Δηλώνουμε και αρχικοποιούμε το ***ps***, μία αναφορική μεταβλητή τύπου ***PrintStream*** και αντιστοιχούμε σε ένα αρχείο
- Στη συνέχεια γράφουμε στο standard output, όπως με την System.out, με τις print(), println()
- **Ο PrintStream μπορεί να γράφει και text και bytes**
- Χρησιμοποιείται βασικά για text, γιατί για bytes υπάρχουν πιο efficient κλάσεις

Generic Print Stream Method


```java
package gr.aueb.cf.cf9.ch9;

import java.io.FileNotFoundException;
import java.io.PrintStream;

public class PrintStreamMethod {

    public static void main(String[] args) throws FileNotFoundException {
        PrintStream ps = new PrintStream("C:/tmp/cf.txt");

        printMsg(ps, "Hello Coding Plus");          // Prints to ps
        printMsg(System.out, "Coding Factory");     // Prints to stdout
    }

    /**
     * Prints a string message to PrintStream.
     *
     * @param ps        the PrintStream object
     * @param message   the message to print
     */
    public static void printMsg(PrintStream ps, String message) {
        ps.println(message);
    }
}
```

- Ορίζουμε μία γενική printMsg() που λαμβάνει ως παράμετρο τον τύπο του PrintStream καθώς και το message
- Παρατηρούμε στην main() την ευελιξία στην κλήση της printMsg() είτε με ps ή με System.out, όπου και τα δύο είναι τύπου PrintStream

PrintWriter


```
public class File3FileWriterPrintStream8 {

    public static void main(String[] args) {
        File fd = new File("C:/tmp/file-writer.txt");
        try {
            bufferedWriter(fd);
        } catch (IOException e) {
            System.out.println("Το αρχείο δεν δημιουργήθηκε. ");
        }
    }

    public static void printWriter(String file) throws IOException {
        try (PrintWriter ps = new PrintWriter(file)) {
            ps.println("Using PrintWriter");
            ps.flush();
        } catch (IOException e) {
            System.out.println(LocalDateTime.now() + "\n" + e);
            throw e;
        }
    }
```

- Παρόμοιο με τον PrintStream αλλά μόνο για text

Αρχεία Χαρακτήρων με Scanner και PrintWriter

- Όταν θέλουμε να γράφουμε τύπους δεδομένων σε αρχεία ιδιαίτερα formatted (με printf()) μπορούμε να χρησιμοποιούμε ***PrintWriter ή PrintStream***
- Για large file writes χρησιμοποιούμε **BufferedWriter**
- Ο FileWriter δεν είναι efficient γιατί γράφει char-by-char. Το κάνουμε wrap σε **BufferedWriter**

Encoding – Charsets (1)

- Υπάρχουν τα εξής βασικά συστήματα απεικόνισης χαρακτήρων:
  - Το κλασικό ***ASCII*** που αντιστοιχεί 128 λατινικούς χαρακτήρες σε 7-bit αριθμούς
  - Τα συστήματα ***ISO-8859-x***, που προσθέτουν 1 ακόμα bit (συνολικά 8-bit) ώστε να μπορούν να απεικονίζουν επιπρόσθετα 128 χαρακτήρες άλλης γλώσσας (πέραν της Αγγλικής). Παραδείγματα αποτελούν τα ***ISO-8859-1 (Latin1 – Western European Γλώσσες)*** ή ***ISO-8859-7 (Latin-Greek – Υποστηρίζουν και Ελληνικά)***. Είναι συμβατά με ASCII
  - Τα συστήματα ***Unicode***, όπως ***UTF-8*** που χρησιμοποιεί 1-4 bytes (μεταβλητός αριθμός bits) και απεικονίζει τους χαρακτήρες όλων των γλωσσών. Είναι συμβατά με ASCII
  - Το ***Windows-1252 (CP-1252)*** που είναι παρόμοιο με το ISO-8859-1 καθώς και το ***CP-1253*** που είναι παρόμοιο με το ISO-8859-7. Συμβατά με ASCII

Encoding – Charsets (2)

- Όλοι οι writers μας δίνουν τη δυνατότητα να επιλέξουμε charset, ώστε η κωδικοποίηση των χαρακτήρων στα αρχεία που γράφουμε να είναι περισσότερο ή λιγότερο συμβατή με άλλα συστήματα
- Αν δεν ορίσουμε κωδικοποίηση, χρησιμοποιείται το default charset του συστήματος (μέχρι την Java 17, **από την Java 18 και μετά χρησιμοποιείται UTF-8 ως default**)
- Αν χρησιμοποιούμε Ελληνικά, ή Windows-1252 και ISO-8859-1 (Latin1) δεν είναι συμβατές κωδικοποιήσεις, ενώ οι Windows-1253, ISO-8859-7 (Latin-Greek) και UTF-8 υποστηρίζουν Ελληνικά
- Επίσης, αν θέλουμε να υποστηρίζουμε Ελληνικά αλλά και να είμαστε συμβατοί όχι μόνο με Windows, αλλά και με άλλα συστήματα τότε χρησιμοποιούμε UTF-8

Create vs Append σε αρχεία

- Σε ένα αρχείο μπορούμε κάθε φορά που ανοίγουμε για εγγραφή ***να γράφουμε είτε από την αρχή***, να δημιουργείται δηλ. το αρχείο (create) ***ή να προσθέτουμε στο τέλος*** (append)
- Το default είναι να δημιουργείται το αρχείο κάθε φορά που συνδεόμαστε με ένα αρχείο και γράφουμε
- Αν θέλουμε ***append ορίζουμε ως true μία παράμετρο του FileOutputStream*** (βλ. επόμενη διαφάνεια)

Append και Charset

- Οι FileWriter, PrintWriter, PrintStream και BufferedWriter έχουν εκδόσεις που υποστηρίζουν charset και append mode
- Μπορούν να αρχικοποιηθούν και με το charset, ώστε το αρχείο που θα δημιουργηθεί να μην έχει το default charset του συστήματος (π.χ. Windows-1252 –Latin - ή Windows-1253 -Greek-). Αυτό που ορίζουμε, μέσω της σταθεράς StandardCharsets.UTF_8
- Επίσης με true ορίζουμε ότι τα αρχεία ανοίγουν για Append. Το false στο PrintStream αφορά το auto-flush

```
try (BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(file, true),
        StandardCharsets.UTF_8))) {
```

```
try (PrintStream ps = new PrintStream(new FileOutputStream(file, true), false, StandardCharsets.UTF_8)) {
```

```
try (FileWriter fw = new FileWriter(file, StandardCharsets.UTF_8, true)) {
```

Σύγκριση writers

Σύγκριση σε πίνακα

Κλάση	Type	Buffered	Exceptions	Formatted output	Use case
FileWriter	Character	✗	checked IOException	✗	Small files, simple char writes
BufferedWriter	Character	✓	checked IOException	✗	Large text files, efficiency
PrintStream	Byte	✗	Δεν έχει checked	✓ (`print/println`)	Logging, debugging, ASCII files
PrintWriter	Character	✗	Δεν έχει checked	✓ (`print/println/printf`)	Text files, formatted output

- Για ASCII files μπορούμε να χρησιμοποιούμε PrintStream ή PrintWriter όπου δεν χρειάζεται και διαχείριση exceptions
- Αν πρόκειται για μεγάλα text αρχεία είναι προτιμότερο να χρησιμοποιούμε BufferedWriter

FileReader


```java
/**
 * Char-by-char read. Low performance.
 * @param file
 * @throws IOException
 */
public static void fileReaderRead(String file) throws IOException {

    // try with resources
    try (FileReader fr = new FileReader(file)) {
        int c;
        while ((c = fr.read()) != -1) {
            System.out.print((char) c);
        }
    } catch (IOException e) {
        System.err.println(LocalDateTime.now() + "fileReaderRead" + e);
    }
//      No need for finally if we try-with-resources
//      finally {
//          try {
//              fr.close();
//          } catch (Exception e) {
//              System.out.println(e);
//          }
//      }
}
```

- Ο FileReader **διαβάζει char-by-char** και δεν έχει καλό performance όπως και ο FileWriter

Buffered reader


```java
// for large texts, powerful read-line
public static void bufferedReaderRead(String file) throws IOException {

    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        String line;

        // Powerful read-line
        while ((line = br.readLine()) != null) {
            System.out.println(line);
        }
    } catch (IOException e) {
        System.out.println(LocalDateTime.now() + "," + e);
        throw e;
    }
}
```

- Ο BufferedReader είναι wrapper του FileReader. Το **default buffer size είναι 8192 bytes**. Είναι efficient για large files

Scanner


```java
// For structured text with primitives - > int double string structured
public static void scannerRead(String file) throws FileNotFoundException {
    try (Scanner sc = new Scanner(new File(file))) {
        while (sc.hasNextLine()) {
            System.out.println(sc.nextLine());
        }
    } catch (FileNotFoundException e) {
        System.out.println(LocalDateTime.now() + "," + e);
        throw e;
    }
}
```

- Όταν το input text file είναι δομημένο, ο Scanner είναι καλύτερος να χρησιμοποιείται γιατί παρέχει build-in μεθόδους για να διαβάζει primitive ή String tokens

Append και Charset

- Όπως για το write, το ίδιο ισχύει και για το read όσο αφορά το charset. Αν δεν ορίσουμε, το default charset για read είναι το charset του Λειτουργικού Συστήματος

```
BufferedReader  new BufferedReader(new InputStreamReader(fis, StandardCharsets.UTF_8)

Scanner new Scanner(new FileInputStream(path), StandardCharsets.UTF_8)
```

Σύγκριση readers

Εργαλείο	Για τι είναι;	Πότε το χρησιμοποιούμε;
FileReader	Διαβάζει χαρακτήρες χωρίς buffer	Σπάνια, μόνο για raw low-level reading
BufferedReader	Γρήγορη ανάγνωση text + readLine()	**Κορυφαία επιλογή** για μεγάλα text files
Scanner	Parsers για ints, doubles, tokens	Input από χρήστη, parsing μικρών αρχείων

- Ο **Scanner** είναι ιδανική επιλογή όταν το input file έχει primitives
- Ο **BufferedReader** είναι ιδανική επιλογή όταν έχουμε μεγάλο text file
- Ο **FileReader** δεν αποτελεί επιλογή γιατί διαβάζει ένα-ένα τους χαρακτήρες

Παράδειγμα 1 (1)

- Έστω ότι έχουμε ένα αρχείο ονομάτων και πόλεων

names.txt - Σημειωματάριο

Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια

```
Αθανάσιος,      Ανδρούτσος,     Αθήνα
Νίκη,           Γιαννούτσου,    Λαμία
Ανδρέας,        Βερούσης,       Πρέβεζα
```

- Και θέλουμε να παράγουμε ένα αρχείο εξόδου

names-formatted.txt - Σημειωματάριο

Αρχείο Επεξεργασία Μορφή Προβολή Βοήθεια

```
{ "Firstname": "Αθανάσιος" , "Lastname": "Ανδρούτσος", "City": "Αθήνα" },
{ "Firstname": "Νίκη" , "Lastname": "Γιαννούτσου", "City": "Λαμία" },
{ "Firstname": "Ανδρέας" , "Lastname": "Βερούσης", "City": "Πρέβεζα" },
```

Παράδειγμα 1 (2)


```java
package gr.aueb.cf.cf9.ch9;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;
import java.nio.charset.StandardCharsets;

public class NamesScanning {

    public static void main(String[] args) {
        String inFilePath = "C:/tmp/names.txt";
        String outFilePath = "C:/tmp/names-formatted.txt";

        try (BufferedReader reader = new BufferedReader(new FileReader(inFilePath));
             PrintStream ps = new PrintStream(outFilePath, StandardCharsets.UTF_8)) {

            String line;
            String[] tokens;
```

- Θα διαβάζουμε με BufferedReader line-by-line και θα γράφουμε με PrintStream

- Θα κάνουμε split κάθε line στα επιμέρους tokens (substrings όπως όνομα, επώνυμο, πόλη) στο String[]

Παράδειγμα 1 (3)


```
21              while ((line = reader.readLine()) != null) {
22                  tokens = line.split(",+\\s*");
23                  ps.printf("{ \"Firstname\": \"%s\"', \"Lastname\": \"%s\", \"City\": \"%s\" },\n", tokens[0], tokens[1], tokens[2]);
24                  System.out.printf("{ \"Firstname\": \"%s\"', \"Lastname\": \"%s\", \"City\": \"%s\" },\n", tokens[0], tokens[1], tokens[2]);
25              }
26          } catch (IOException e) {
27              System.err.println("Error reading the file: " + e.getMessage());
28          }
29      }
```

- Επειδή δεν ξέρουμε κάθε αρχείο πόσες lines έχει, τυπικά διαβάζουμε μέσα σε μία for. Όσο το readLine() είναι διάφορο του null. Null είναι το end-of-file. Το *,+\\s** είναι regular expression (pattern recognizer) και σημαίνει ένα ή περισσότερα , ακολουθούμενα από 0 ή περισσότερα whitespaces. Θα δούμε σε επόμενα μαθήματα regular expressions και ποσοδείκτες
- Γράφουμε στο **ps** που είναι το PrintStream καθώς και στο Standard output
- Επειδή οι πράξεις σε αρχεία μπορεί να δώσουν *FileNotFoundException* και *IOException* που είναι υπερκλάση του *FileNotFoundException* μπορούμε απλά να κάνουμε catch το *IOException* (που συμπεριλαμβάνει και το *IOException*)

Παράδειγμα 1 (4)


```
31 @    public static void printFormatted(PrintStream ps, String[] tokens) {
32          ps.printf("{ \"Firstname\": \"%s\"', \"Lastname\": \"%s\", \"City\": \"%s\" },\n", tokens[0], tokens[1], tokens[2]);
33      }
```

- Εναλλακτικά μπορεί να έχουμε μία γενική μέθοδο που παίρνει δύο παραμέτρους, ένα PrintStream και ένα πίνακα String[] και εκτυπώνει (βλ. επόμενη διαφάνεια)

Παράδειγμα 1 (5)


```
15          try (BufferedReader reader = new BufferedReader(new FileReader(inFilePath));
16               PrintStream ps = new PrintStream(outFilePath, StandardCharsets.UTF_8)) {
17
18              String line;
19              String[] tokens;
20
21              while ((line = reader.readLine()) != null) {
22                  tokens = line.split(",+\\s*");
23  //                ps.printf("{ \"Firstname\": \"%s\"', \"Lastname\": \"%s\", \"City\": \"%s\" },\n", tokens[0], tokens[1], to
24  //                System.out.printf("{ \"Firstname\": \"%s\"', \"Lastname\": \"%s\", \"City\": \"%s\" },\n", tokens[0], token
25                  printFormatted(ps, tokens);
26                  printFormatted(System.out, tokens);
27              }
28          } catch (IOException e) {
29              System.err.println("Error reading the file: " + e.getMessage());
30          }
31      }
32
33  @   public static void printFormatted(PrintStream ps, String[] tokens) {
34          ps.printf("{ \"Firstname\": \"%s\"', \"Lastname\": \"%s\", \"City\": \"%s\" },\n", tokens[0], tokens[1], tokens[2]);
35      }
36  }
```

- Στις γραμμές 25, 26 κάνουμε invoke τις μεθόδους με ps και System.out (το System.out είναι PrintStream)

BufferedReader και java.io


```
while ((line = reader.readLine()) != null) {
```

- Παρατηρούμε ότι αφού δηλώσουμε ένα ***String line*** στη συνέχεια -για να διαβάσουμε από τον ***BufferedReader***- μέσα σε μια while διαβάζουμε μέχρι να βρούμε τέλος αρχείου, δηλ. λέμε ***«όσο δεν έχεις βρει τέλος αρχείου, διάβαζε την επόμενη γραμμή».***
- Αν η ***readLine()*** βρει τέλος αρχείου επιστρέφει ***null.*** Άρα ***διαβάζουμε όσο το line δεν είναι null***
- Παρατηρούμε επίσης ότι δεν γράφουμε (line != null) αλλά ***(line = readLine()) != null*** , δηλαδή ***διαβάζουμε και συγκρίνουμε***. Αυτό που συγκρίνουμε είναι μία παράσταση μέσα σε παρενθέσεις: (line = readLine())
- Στην πραγματικότητα ***η τιμή μιας παράστασης είναι ή τιμή του αριστερού μέλους***, δηλαδή της line, άρα τελικά την line συγκρίνουμε με το null
- Αυτό το στυλ κώδικα είναι παρμένο από τη γλώσσα προγραμματισμού C

Παράδειγμα 2 – GeoData (1)

- Η getGeoData(), διαβάζει γεωχωρικά δεδομένα από ένα αρχείο και τα επιστρέφει σε μορφή πίνακα με τέσσερις στήλες
- Η main λαμβάνει από την getGeoData() και εκτυπώνει τα data

id,name,latitude,longitude
1,Acropolis,37.971532,23.725749
2,White Tower,40.626389,22.948611
3,Olympus Mountain,40.085556,22.358333

Παράδειγμα 2 – GeoData (2)


```java
    public static String[][] getGeoData(File file) throws FileNotFoundException {
        int id = 0;
        String name = null;
        double latitude = 0.0;
        double longitude = 0.0;
        String line = null;
        String[] parts = null;
        int count = -1;
        String[][] returnedArray = new String[1000][4];

        try (Scanner scanner = new Scanner(file)) {

            // Skip header line
            if (scanner.hasNextLine()) {
                scanner.nextLine();
            }

            // Read each geo-record
            while (scanner.hasNextLine()) {
                count++;
                line = scanner.nextLine();
                parts = line.split(",");

                id = Integer.parseInt(parts[0]);
                name = parts[1];
                latitude = Double.parseDouble(parts[2]);
                longitude = Double.parseDouble(parts[3]);

                returnedArray[count]  = parts;
            }
            return Arrays.copyOf(returnedArray, count) ;
        } catch (FileNotFoundException e) {
            System.err.println(e.getMessage());
            throw e;
        }
    }
}
```

```java
package gr.aueb.cf.cf9.ch9;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Arrays;
import java.util.Scanner;

public class GeoReader {
    public static void main(String[] args) {
        File file = new File("C:/tmp/locations.csv");
        String[][] geoData;

        try {
            geoData = getGeoData(file);
            for (String[] geoDatum : geoData) {
                for (String part : geoDatum) {
                    System.out.print(part + " ");
                }
                System.out.println();
            }
        } catch (FileNotFoundException e) {
            System.out.println("Το αρχείο δεν βρέθηκε: " + e.getMessage());
        }
    }
```

- Η getGeoData(), διαβάζει γεωχωρικά δεδομένα από ένα αρχείο και τα επιστρέφει σε μορφή πίνακα
- Η main λαμβάνει από την getGeoData() και εκτυπώνει τα data

Scanner vs BufferedReader (1)

- Ας υποθέσουμε ότι ***θέλουμε να διαβάζουμε ολόκληρες γραμμές χαρακτήρων η ακόμα μεγαλύτερο πλήθος χαρακτήρων με ένα μόνο read***.
- Ο Scanner δεν είναι efficient γιατί ενώ το default block size που διαβάζουμε από το δίσκο είναι 4KB-16KB στα Windows/MacOS και 4KB-8KB σε UNIX/Linux συστήματα, το buffer size του Scanner είναι μόνο 1KB

Scanner vs BufferedReader (2)

- Ο ***BufferedReader*** χρησιμοποιεί ένα εσωτερικό buffer (περιοχή της μνήμης) με μήκος 8192 (=8KB) bytes δηλ. Χαρακτήρες (ενώ ο Scanner όπως αναφέραμε μόνο 1KB)
- Αν πρόκειται να διαβάζουμε γραμμές, τότε μπορούμε είτε με Scanner και την nextLine() ή με ***BufferedReader*** και την ***readLine()***

```
while ((line = br.readLine()) != null) {
    System.out.println(line);
}
```

- Και ο Scanner και ο BufferedReader καταναλώνουν την new line, αλλά δεν την αποδίδουν στο επιστρεφόμενο String

Scanner vs BufferedReader (3)

- Σε αντίθεση με τον Scanner, ο BufferedReader/Writer είναι:
  - ***Synchronized*** -- κλειδώνει (locks) όταν κάποιος διαβάζει και δεν επιτρέπεται ταυτόχρονη επεξεργασία από διάφορα threads, άρα είναι thread-safe
  - Μπορεί να επιλέξει το μέγεθος του buffer και να διαβάζει με ένα read μεγάλο πλήθος χαρακτήρων, π.χ. 8192 (8KB)
  - Διαβάζει γρηγορότερα από τον Scanner
  - Δημιουργεί ***IOException*** (ενώ ο Scanner *FileNotFoundException*)

Αρχεία αμορφοποίητων δεδομένων - Ροές Bytes

- Μπορούμε να **διαβάζουμε ροές bytes** (π.χ. αρχεία εικόνων ή βίντεο) με την ***FileInputStream***
- Μπορούμε να **γράφουμε ροές bytes** με την ***FileOutputStream***
- *Ροές bytes είναι και αρχεία κειμένου όπως pdf τα οποία όμως δεν είναι απλό κείμενο αλλά με ειδική μορφοποίηση, οπότε και τα θεωρούμε binary files*

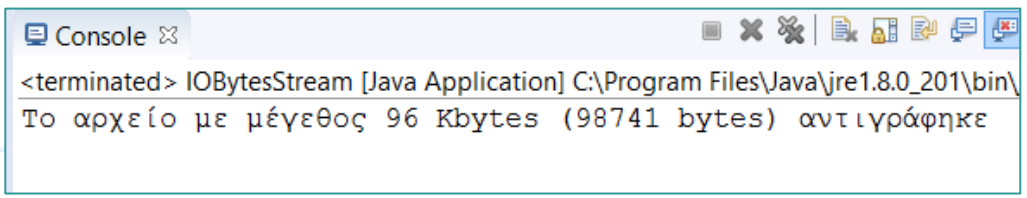
Αντιγραφή εικόνας (1)

- Στο επόμενο παράδειγμα θα δούμε πως ***αντιγράφουμε ένα αρχείο εικόνας byte-byte και ταυτόχρονα υπολογίζουμε το μέγεθός του σε ψηφιοσυλλαβές (bytes)*** χρησιμοποιώντας ***FileInputStream*** και ***FileOutputStream***

Αντιγραφή εικόνας (2)


```
1  package gr.aueb.cf.cf9.ch9;
2
3  import java.io.FileInputStream;
4  import java.io.FileOutputStream;
5  import java.io.IOException;
6
7  /**
8   * Το πρόγραμμα αυτό αντιγράφει ένα αρχείο εικόνας
9   * και ταυτόχρονα υπολογίζει το μέγεθος του αρχείου
10  * που αντέγραψε.
11  *
12  * @author thanos
13  *
14  */
15 public class IOBytesStream {
16
17     public static void main(String[] args) {
18
19         try(FileInputStream in = new FileInputStream("C:/Users/thanos/jtmp/image1.jpeg");
20                 FileOutputStream out = new FileOutputStream("C:/Users/thanos/jtmp/image1Copy.jpeg");)
21         {
22             int b, count=0;
23             while ((b = in.read()) != -1) {
24                 out.write(b);
25                 count++;
26             }
27             System.out.printf("Το αρχείο με μέγεθος %d Kbytes (%d bytes) αντιγράφηκε", count/1024, count);
28
29         }catch (IOException e) {
30             System.out.println(e.getMessage());
31         }
32     }
33 }
```

- Παρατηρούμε ότι ***η read() διαβάζει ένα byte τη φορά*** (τα bytes είναι int) μέχρι να μη βρει τίποτα (-1)
- Κάθε φορά γράφει το byte στο out και αυξάνει τον μετρητή των bytes κατά 1



Buffering (1)

- Η αντιγραφή ένα-ένα byte ***δεν είναι αποδοτική*** γιατί γίνονται τόσα read και write όσα και τα bytes του αρχείου και κάθε read είναι ένα system call, μία αναφορά δηλ. στον σκληρό δίσκο ή SSD
- Θα θέλαμε ***με ένα read να διαβάζουμε περισσότερα bytes (buffering)*** ώστε να μειώσουμε τα system calls που είναι ακριβά σε όρους χρόνου
- Έχει λοιπόν σημασία η πολιτική buffering που ακολουθούμε, δηλαδή πόσα bytes διαβάζουμε και γράφουμε τη φορά

Buffering(2)

- Οι βασικές επιλογές είναι **FileInputStream/FileOutputStream** με read σε buffer και
- ***BufferedInputStream/BufferedOutputStream*** που κάνει αυτόματο buffering 8192 bytes (8KB)

Αντιγραφή byte προς byte


```java
package gr.aueb.cf.cf9.ch9;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * Αντιγράφει ένα pdf αρχείο (ως binary file)
 * αντιγράφοντας 1 byte τη φορά
 */
public class PdfCopyApp {

    public static void main(String[] args) {

        try (FileInputStream fis = new FileInputStream("C:/tmp/museum-in.pdf");
             FileOutputStream fos = new FileOutputStream("C:/tmp/museum-out.pdf")) {

            int b;
            int count = 0;
            long start;
            long end;
            double elapsedTime = 0.0;

            start = System.currentTimeMillis();
            while ((b = fis.read()) != -1) {
                fos.write(b);
                count++;
            }
            end = System.currentTimeMillis();
            elapsedTime = (end - start) / 1000.0;

            System.out.printf("Το αρχείο με μέγεθος %dKB (%d bytes) αντιγράφηκε επιτυχώς",
                    (count / 1024), count);
            System.out.println("Elapsed time: " + elapsedTime + " seconds");

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- Διαβάζουμε με τη read() ένα byte τη φορά και γράφουμε με τη write(b) το 1 byte που διαβάσαμε
- Αυξάνουμε τον counter κατά 1
- Ο χρόνος εμφανίζεται σε sec και βλέπουμε την τιμή, πολύ υψηλή

Αντιγραφή με buffer


```java
package gr.aueb.cf.cf9.ch9;

import java.io.*;

public class PdfBufferedCopyApp {

    public static void main(String[] args) {
        try (BufferedInputStream bis = new BufferedInputStream(new FileInputStream("C:/tmp/museum-in.pdf"));
             BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("C:/tmp/museum-out.pdf")) ) {

            int b;
            int count = 0;
            long start;
            long end;
            double elapsedTime = 0.0;
            byte[] buf = new byte[4096];

            start = System.currentTimeMillis();
            while ((b = bis.read(buf)) != -1) {
                bos.write(buf, 0, b);
                count += b;
            }
            end = System.currentTimeMillis();
            elapsedTime = (end - start) / 1000.0;

            System.out.printf("Το αρχείο με μέγεθος %dKB (%d bytes) αντιγράφηκε επιτυχώς", (count / 1024), count);
            System.out.println("Elapsed time: " + elapsedTime + " seconds");

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- Ο ***χρόνος αντιγραφής είναι πολύ μικρότερος*** από ότι αν κάναμε την αντιγραφή byte-byte και αυτό οφείλεται στο ότι κάνουμε λιγότερα read/write αφού ο buffer έχει μήκος 8192
- Οι read/write παίρνουν ως παράμετρο το ***buf, που είναι ένας πίνακας bytes μήκους 8192***

```
"C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe" "-javaagent:C:\Program Files\Jet
Το αρχείο με μέγεθος 2919KB (2989354 bytes) αντιγράφηκε επιτυχώςElapsed time: 0.007 seconds

Process finished with exit code 0
```

NIO (New IO) - Paths

- interface **Path**
  - Το **Path** είναι container ενός μονοπατιού (path)
- **Files** (utility class)
  - Παρέχει μεθόδους για read, write, copy, move, delete, κλπ.

NIO (New IO) – Files (1)

Operation	NIO (`Files`)	Classic IO (`java.io`)
Check if file exists	`Files.exists(path)`	`file.exists()`
Create directories	`Files.createDirectories(path)`	`file.mkdirs()`
Read all bytes	`byte[] data = Files.readAllBytes(path)`	`FileInputStream fis = new FileInputStream(file); fis.read(bytes);` (loop may be needed)
Read all lines	`List<String> lines = Files.readAllLines(path)`	`BufferedReader br = new BufferedReader(new FileReader(file)); String line; while((line = br.readLine()) != null)`
Write string	`Files.writeString(path, "Hello")`	`FileWriter fw = new FileWriter(file); fw.write("Hello"); fw.close();`
Write bytes	`Files.write(path, byteArray)`	`FileOutputStream fos = new FileOutputStream(file); fos.write(byteArray); fos.close();`
Copy file	`Files.copy(source, target)`	`FileInputStream fis = new FileInputStream(src); FileOutputStream fos = new FileOutputStream(dest); copy loop`
Move / rename file	`Files.move(source, target, StandardCopyOption.REPLACE_EXISTING)`	`file.renameTo(destFile)` (no overwrite control, may fail silently)
Delete file	`Files.delete(path)`	`file.delete()`
Stream lines / functional processing	`Stream<String> lines = Files.lines(path)`	Not available; would need manual `BufferedReader` loop
Get file attributes	`Files.size(path)` / `Files.isDirectory(path)`	`file.length()` / `file.isDirectory()`

- Στην στήλη της NIO εμφανίζονται οι στατικές μέθοδοι της κλάσης **Files**, με τη χρήση **Path** που υποστηρίζουν τόσο εγγραφή/ανάγνωση κειμένου και bytes, όσο και δημιουργία φακέλων, αντιγραφή, μετακίνηση ή διαγραφή αρχείων, καθώς και λειτουργίες streaming με χαμηλή κατανάλωση μνήμης.
- Η στήλη της κλασικής Java IO δείχνει τις αντίστοιχες λειτουργίες χρησιμοποιώντας File.
- Η σύγχρονη NIO είναι πιο ευέλικτη, πιο ασφαλής και κατάλληλη για μεγάλα αρχεία ή εφαρμογές που χρειάζονται ροές δεδομένων και λειτουργικότητα με Unicode.

NIO (New IO) – Files (2)

Get absolute path	`file.getAbsolutePath()`	`path.toAbsolutePath()`
Get file name	`file.getName()`	`path.getFileName()`
Get parent directory	`file.getParent()`	`path.getParent()`

- Επίσης παρέχονται μέθοδοι για να επιστρέφεται το όνομα ενός path είτε το απλό ή το πλήρες ή το parent
- π.χ. στο /home/pages/index.html
- Το index.html είναι το name
- Το /home/pages/index.html είναι το πλήρες name
- Το /home/pages είναι το parent

NIO text writer


```java
public class NioReadAndWrite {

    public static void main(String[] args) throws IOException {
        Path path = Path.of("C:/tmp/test.txt");
        textWriter(path);
    }

    public static void textWriter(Path path) throws IOException {
        Files.createDirectories(path.getParent());  // creates the directory if it doesn't exist
        Files.writeString(
                path,
                "Hello",
                StandardCharsets.UTF_8,
                StandardOpenOption.CREATE,
                StandardOpenOption.APPEND
        );
    }
```

- Με Files.createDirectories(path.getParent() διασφαλίζεται ότι **οι φάκελοι που οδηγούν στο αρχείο υπάρχουν**. Αν δεν υπάρχουν, δημιουργούνται αυτόματα. Αν ήδη υπάρχουν, δεν συμβαίνει τίποτα.
- Στη συνέχεια εκτελεί την **εγγραφή της λέξης "Hello" στο αρχείο**.
  - StandardCharsets.UTF_8 εξασφαλίζει ότι το κείμενο θα γραφτεί σε UTF-8.
  - StandardOpenOption.CREATE δημιουργεί το αρχείο αν δεν υπάρχει.
  - StandardOpenOption.APPEND προσθέτει το νέο κείμενο στο τέλος του αρχείου αντί να το αντικαθιστά

NIO Text reader


```java
/**
 * Closes automatically. No need for try-with-resources.
 * @param path
 * @throws IOException
 */
public static void textReader(Path path) throws IOException {
    String text = Files.readString(path, StandardCharsets.UTF_8);
    System.out.println(text);
}

public static void textReaderByLine(Path path) throws IOException {
    try (var lines = Files.lines(path, StandardCharsets.UTF_8)) {
        lines.forEach(System.out::println);
    }
}
```

- Με readString() διαβάζουμε από ένα Path και επιστρέφουμε String
- Αν είναι μεγάλο το αρχείο μπορούμε με **Files.lines()** να διαβάζουμε γραμμή-γραμμή. Επιστρέφει stream και μπορούμε και εκτυπώνουμε ευέλικτα με forEach() (θα δούμε τη μορφή αυτή σε επόμενα κεφάλαια)

Αντιγραφή binary αρχείου


```java
public static void binaryWriter(Path source, Path target) {
    try (var sourceStream = Files.newInputStream(source);
         var targetStream = Files.newOutputStream(target)) {
        sourceStream.transferTo(targetStream);  // efficient, streams in chunks
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

- Το NIO παρέχει την **transferTo()** που είναι πολύ ευέλικτη και αντιγράφει *lazily* (δεν αποθηκεύει όλο το file στη μνήμη *eagerly*, αλλά *lazily* chunk-by-chunk)