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 Mac 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\a8ana ενώ στα Linux-based συστήματα το home dir είναι /home/a8ana και /Users/a8ana σε macOS
- Το home folder/home dir συμβολίζεται με ~
- 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 φακέλου από τη ρίζα ονομάζεται μονοπάτι (***path***)
- Για παράδειγμα το ***C:\Users\a8ana\docs*** είναι ένα μονοπάτι (path)
- Το πλήρες όνομα (Fully Qualified File Name) ενός αρχείου είναι το μονοπάτι μαζί με το όνομα αρχείου. Για παράδειγμα, στα Windows: ***C:\Users\thanos\docs\tmp.doc***
- Στα Linux ***/home/a8ana/docs/tmp.doc***
- *Σε macOS* ***/Users/a8ana/docs/tmp.doc***

Java I/O

- Το I/O στην Java 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
  - Write

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 (Non-blocking 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)

Αρχεία

- Το κλασικό Java I/O μας δίνει την κλάση **File** για να αναπαριστούμε αρχεία
- Παρακάτω 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

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

File & Scanner


```java
package testbed.ch9;

import java.io.File;
import java.io.IOException;
import java.util.Scanner;

public class FileApp {

    public static void main(String[] args) {
        File fd = new File("C:/tmp/file6.txt");

        try (Scanner in = new Scanner(fd)) {
            while (in.hasNextLine()) {
                String line = in.nextLine();
                String[] tokens = line.split(" ");

                for (String token : tokens) {
                    System.out.printf("%s ", token);
                }
                System.out.println();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

- Σχετίζουμε το File fd με ένα path και στη συνέχεια το αντιστοιχούμε σε ένα Scanner για να διαβάζουμε από αρχείο που έχει primitives

Αρχεία και ροές εισόδου-εξόδου

- Στον προγραμματισμό οτιδήποτε εισέρχεται στο σύστημα είναι ροή εισόδου (***input stream***) και οτιδήποτε εξέρχεται είναι ροή εξόδου (***output stream***)
- Όλες οι ροές (streams) σχετίζονται με αρχεία (files) (ή με δικτυακές ροές)
- Οι ροές λειτουργούν ως κανάλια μεταφοράς δεδομένων όπως row data (*bytes)*, characters, *primitives*, objects
- Είδαμε πως μπορούμε με Scanner να διαβάζουμε από text file που περιείχε primitives αλλά και strings

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 αμορφοποίητων δεδομένων **BufferedOutputStream** -- Χρησιμοποιεί εσωτερικό buffer
Αρχεία Κειμένου	***Scanner*** ***BufferedReader / InputStreamReader / FileReader*** -- Χρησιμοποιούνται για είσοδο δεδομένων χαρακτήρων ***PrintStream*** ***PrintWriter / BufferedWriter / OutputStreamWriter / FileWriter*** -- 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

Είσοδος / Έξοδος δεδομένων

- ***Scanner, PrintWriter.*** Μέχρι στιγμής έχουμε δει την είσοδο δεδομένων από το πληκτρολόγιο και από αρχείο με τη χρήση Scanner και την έξοδο αποτελεσμάτων στο sdout με τη χρήση PrintStream (το System.out είναι PrintStream). Ωστόσο το PrintStream χειρίζεται κυρίως row bytes, ενώ ο **PrintWriter** έχει σχεδιαστεί για character streams και μπορεί να χειριστεί το encoding ευκολότερα
- **InputStream, OutputStream.** Παρέχουν τις βασικές μεθόδους, read(), write() που διαβάζουν, γράφουν ένα byte τη φορά

Java I/O

- Το java.io package παρέχει έτοιμες I/O κλάσεις και μεθόδους για είσοδο/έξοδο δεδομένων
- Όπως είπαμε υπάρχουν δύο είδη δεδομένων και αντίστοιχων ροών
  - 1. Character-oriented
    - Χρησιμοποιεί ροές δεδομένων τύπου χαρακτήρα
  - 2. Byte streams
    - Χρησιμοποιεί ροές αμορφοποίητων δεδομένων bytes

Αρχεία Χαρακτήρων με Scanner και PrintWriter

- Όταν θέλουμε να διαβάζουμε τύπους δεδομένων από αρχεία (όπως χαρακτήρες / ακεραίους / δεκαδικούς, Strings κλπ.) μπορούμε να χρησιμοποιούμε ***Scanner***
- Επίσης, όταν θέλουμε να γράφουμε τύπους δεδομένων σε αρχεία μπορούμε να χρησιμοποιούμε ***PrintWriter ή PrintStream***
- Ο ***PrintStream*** είναι γενικότερος από τον ***PrintWriter***
- Ο **PrintWriter** γράφει μόνο χαρακτήρες ενώ ο **PrintStream** γράφει και χαρακτήρες σαν Writer αλλά και αμορφοποίητα δεδομένα ως γενικό OutputStream

Scanner & PrintWriter


```java
package testbed.ch9;

import ...

public class FileInOut {

    public static void main(String[] args) {
        File fdIn = new File("C:/tmp/file6.txt");
        File fdOut = new File("C:/tmp/file6out.txt");

        try (Scanner in = new Scanner(fdIn);
            PrintWriter pw = new PrintWriter(fdOut, StandardCharsets.UTF_8)) {
            while (in.hasNextLine()) {
                String line = in.nextLine();
                String[] tokens = line.split(" ");

                for (String token : tokens) {
                    pw.printf("%s ", token);
                }
                pw.println();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

- Εφαρμογή που διαβάζει με Scanner και γράφει με printWriter

Εκτύπωση σε αρχείο (1)


```
1  package testbed.ch9;
2
3  import java.io.FileNotFoundException;
4  import java.io.PrintStream;
5
6  /**
7   * Prints with PrintStream class.
8   */
9  public class PrintStreamApp {
10
11     public static void main(String[] args) {
12         try (PrintStream ps = new PrintStream("C:/tmp/f1.txt")) {
13             ps.println("Hello Coding Factory");
14         } catch (FileNotFoundException ex) {
15             ex.printStackTrace();
16         }
17     }
18 }
```

- 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()

Παρατηρούμε πως το path μπορούμε και στα Windows να το δίνουμε με **/** (Θα μπορούσαμε επίσης ως C:\\tmp\\f1.txt)

Generic Print Stream Method


```java
package testbed.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

Εκτύπωση σε αρχείο (2)


```java
package testbed.ch9;

import java.io.FileNotFoundException;
import java.io.PrintWriter;

/**
 * Prints text with PrintWriter.
 */
public class PrintWriterApp {

    public static void main(String[] args) {
        try (PrintWriter pw = new PrintWriter("C:/tmp/fw.txt");) {
            pw.println("Hello World!");
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        }
    }
}
```

- Εκτυπώνουμε χαρακτήρες σε αρχείο με ***PrintWriter***
- Στην Java οι καταλήξεις Writer/Reader σημαίνει διαχείριση αρχείων χαρακτήρες (όχι raw data)

PrintStream & PrintWriter


```java
package gr.aueb.elearn;

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

/**
 * Εκτυπώνει σε αρχείο χαρακτήρες με τη χρήση
 * των PrintStream και PrintWriter.
 *
 * @author A. Androutsos
 */
public class IOFilePrint {

    public static void main(String[] args)  {
        try (PrintStream ps = new PrintStream("C:/Users/thanos/jtmp/test/IOSimpleTest1.txt");
             PrintWriter pw = new PrintWriter("C:/Users/thanos/jtmp/test/IOSimpleTest2.txt"))
        {
            ps.println("Hello World2! from PrintStream");
            pw.println("Hello World2! from PrintWriter");
        } catch (FileNotFoundException e) {
            System.out.println("Το αρχείο δεν βρέθηκε");
        }
    }
}
```

PrintStream με charset


```java
package testbed.ch9;

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

/**
 * Defines PrintStream with charset.
 */
public class PrintStreamCharSetApp {

    public static void main(String[] args) {
        try (PrintStream ps = new PrintStream("C:/tmp/file.txt"
                                    , StandardCharsets.UTF_8);) {
            ps.println("Hello");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- Η κλάση PrintStream μπορεί να αρχικοποιηθεί 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-)
- Αλλά αυτό που ορίζουμε, όπως στο παράδειγμα το UTF-8 μέσω της σταθεράς StandardCharsets.UTF_8

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)

- Οι ***PrintStream*** και ***PrintWriter*** 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, οι 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

Encoding με PrintStream και PrintWriter (1)


```
1  package gr.aueb.cf;
2
3  import java.io.FileNotFoundException;
4  import java.io.PrintStream;
5  import java.io.PrintWriter;
6  import java.io.UnsupportedEncodingException;
7
8  /**
9   * Το πρόγραμμα αυτό δοκιμάζει διάφορα encodings για χρήση με
10  * τους PrintStream και PrintWriter
11  *
12  * Ελληνικά υποστηρίζονται από UTF-8 , Windows-1253 , ISO-8859-7
13  * Ελληνικά ΔΕΝ υποστηρίζονται από Windows-1252 , ISO-8859-1 (Latin1)
14  *
15  * @author thanos
16  *
17  */
18 public class IOEncoding {
19
20     public static void main(String[] args) throws UnsupportedEncodingException, FileNotFoundException {
21
22         PrintStream ps = new PrintStream("C:/Users/thanos/jtmp/testEncode1.txt", "UTF-8");
23         PrintWriter pw = new PrintWriter("C:/Users/thanos/jtmp/testEncode2.txt",  "Windows-1252");
24
25         ps.println("Οικονομικό Πανεπιστήμιο Αθηνών");
26         pw.println("Οικονομικό Πανεπιστήμιο Αθηνών");
27
28         ps.close();
29         pw.close();
30     }
31 }
```

- Δημιουργούμε ένα ***PrintStream ps*** και ένα ***PrintWriter pw*** αντιστοιχώντας στα παρακάτω αρχεία με αντίστοιχη κωδικοποίηση, ***UTF-8*** και ***Windows-1252*** αντίστοιχα

- Αναμένουμε στην 1η περίπτωση με κωδικοποίηση UTF-8 να εμφανίζεται σωστά το κείμενο στα Ελληνικά στα Windows (ή Linux) ενώ στην 2η περίπτωση με Windows-1252 δεν εμφανίζονται τα Ελληνικά (βλ. επόμενη διαφάνεια)

No flush


```java
package testbed.ch9;

import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class FlushApp {

    public static void main(String[] args) {
        try {
            PrintWriter pw = new PrintWriter("C:/tmp/flush-file.txt");
            pw.print("Hello");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

- Δεν γράφει στο stdout, η .print() δεν κάνει flush

Με flush


```java
package testbed.ch9;

import ...

public class FlushApp {

    public static void main(String[] args) {
        try (PrintWriter pw = new PrintWriter("C:/tmp/flush-file.txt");
             Scanner in = new Scanner(System.in)){
            pw.print("Hello");
            pw.flush();
            System.out.println("Insert something to continue");
            in.nextLine();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

- Γράφει στο stdout άμεσα με .flush()

Auto-close και Auto-flush


```java
package testbed.ch9;

import java.io.FileNotFoundException;
import java.io.PrintWriter;

public class FlushApp {

    public static void main(String[] args) {
        try (PrintWriter pw = new PrintWriter("C:/tmp/flush-file.txt");){
            pw.print("Hello");
            //pw.flush();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

- Με try-with-resources γράφει αλλά όχι άμεσα. Μόλις γίνει το auto-close γίνεται auto-flush

Auto-flush με OutputStream


```java
package testbed.ch9;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;

/**
 * Auto-flush enabled with FileOutputStream
 */
public class AutoFlushApp {

    public static void main(String[] args) {
        try (PrintStream ps = new PrintStream(new FileOutputStream("C:/tmp/fos.txt"), true)) {
            ps.println("Hello");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

- Αυτή η μορφή των PrintStream (και PrintWriter το ίδιο) δείχνει ότι πρόκειται για wrapper κλάσεις (κλάσεις περιτυλίγματος), δηλαδή περιτυλίγουν ένα γενικό αρχείο εξόδου (FileOutputStream) και το ‘μετατρέπουν’ σε PrintStream (και PrintWriter αντίστοιχα) με ενεργοποιημένο auto-flush (true) και charset

Auto-flush, charset


```java
package testbed.ch9;

import java.io.*;
import java.nio.charset.StandardCharsets;

/**
 * Auto-flush και charset με FielOutputStream
 */
public class PrintStreamWithFileOutputStreamApp {

    public static void main(String[] args) {
        try (PrintStream ps = new PrintStream(new FileOutputStream("C:/tmp/file1.txt")
                                              , true
                                              , StandardCharsets.UTF_8)) {
            ps.println("Hello");
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
        }
    }
}
```

Auto-flush

- Όσο αφορά το ***PrintStream*** το auto-flush αφορά τα αμορφοποίητα δεδομένα ενώ τα ***text ή character-based δεδομένα εμφανίζονται άμεσα είτε το auto-flush είναι on ή off***
- Όσο αφορά τον ***PrintWriter***, ***αν είναι ενεργοποιημένο το auto-flush, τότε εμφανίζονται άμεσα μόνο δεδομένα που εμπεριέχουν αλλαγή γραμμής*** (π.χ. println()). Αυτό σημαίνει πως όταν εκτυπώνουμε με απλή print() ακόμα κι αν έχουμε ενεργοποιημένο το auto-flush τα δεδομένα δεν εκτυπώνονται άμεσα αλλά παραμένουν στον buffer και εκτυπώνονται όταν κάνουμε close() ή αν κάνουμε ρητά flush()

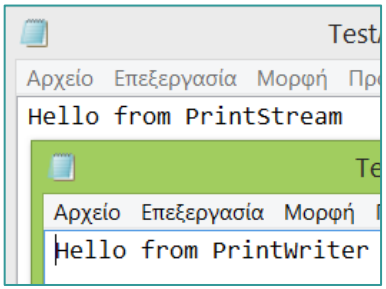
Create vs Append σε αρχεία

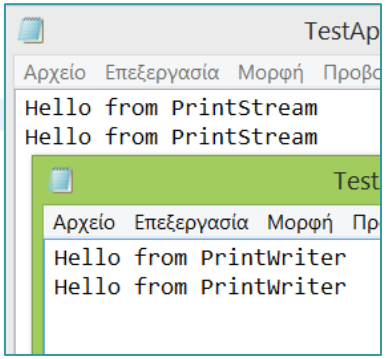
- Σε ένα αρχείο μπορούμε κάθε φορά που ανοίγουμε για εγγραφή ***να γράφουμε είτε από την αρχή***, να δημιουργείται δηλ. το αρχείο (create) ***ή να προσθέτουμε στο τέλος*** (append)
- Το default είναι να δημιουργείται το αρχείο κάθε φορά που συνδεόμαστε με ένα αρχείο και γράφουμε
- Αν θέλουμε ***append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

Προσθήκη στο τέλος (Append)


```java
package gr.aueb.cf;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;
import java.io.PrintWriter;

/**
 * Το πρόγραμμα αυτό επιδεικνύει πως κάνουμε append (δηλ. γράφουμε στο τέλος)
 * σε ένα αρχείο. Παρατηρήστε την FileOutputStream, που παίρνει το όνομα του αρχείου
 * αλλά και μία παράμετρο true που είναι το append
 *
 * Θα τρέξουμε δύο φορές αυτό το πρόγραμμα και
 * αναμένουμε δύο εγγραφές σε κάθε αρχείο
 *
 * @author thanos
 *
 */
public class IOAppend {

    public static void main(String[] args) throws FileNotFoundException {

        PrintStream ps = new PrintStream(new FileOutputStream("C:/Users/thanos/jtmp/TestAppend1.txt", true));
        PrintWriter pw = new PrintWriter(new FileOutputStream("C:/Users/thanos/jtmp/TestAppend2.txt", true));

        ps.println("Hello from PrintStream");
        pw.println("Hello from PrintWriter");


        ps.close();
        pw.close();
    }
}
```

- Στην 1η εκτέλεση δημιουργείται το αρχείο με μία εγγραφή.
- Στην 2η εκτέλεση η εγγραφή γίνεται στο τέλος





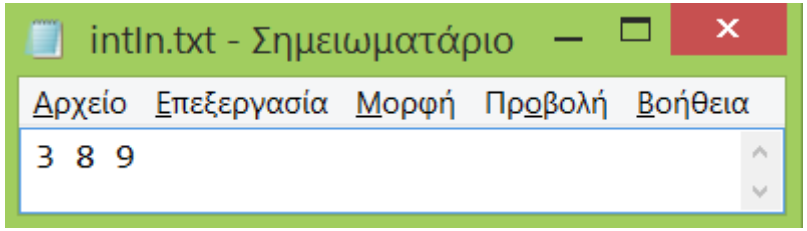
Η 2η παράμετρος στο ***FileOutputStream*** που αντιπροσωπεύει το ***append*** έχει οριστεί ως ***true***

Εφαρμογές

- Για να γίνουν πιο κατανοητά όλα αυτά θα δούμε παρακάτω εφαρμογές
  1. Διάβασμα/επεξεργασία/εμφάνιση αρχείου ακεραίων
  2. Διάβασμα/επεξεργασία/εμφάνιση μικτού αρχείου Strings και ακεραίων

Επεξεργασία αρχείου ακεραίων (1)

- Έστω το παρακάτω αρχείο ακεραίων IntIn.txt:



- Θέλουμε να διαβάσουμε ακέραιους αριθμούς όσο υπάρχει επόμενος και να τους προσθέτουμε και να βρίσκουμε τον μέσο όρο
- Στη συνέχεια να γράφουμε τον μέσο όρο σε ένα αρχείο εξόδου intOut.txt

Επεξεργασία αρχείου ακεραίων (2)

- Δηλώνουμε τα αρχεία εισόδου / εξόδου με Scanner και PrintStream αντίστοιχα

```
1  package gr.aueb.elearn.ch9;
2
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.io.PrintStream;
6  import java.util.Locale;
7  import java.util.Scanner;
8
9  /**
10  * Διαβάζει από ένα αρχείο ακεραίων και γράφει
11  * σε ένα άλλο αρχείο το άθροισμα και τον μέσο όρο.
12  *
13  * @author A. Androutsos
14  */
15 public class IOIntDemo {
16
17     public static void main(String[] args) throws FileNotFoundException {
18         Scanner sc = new Scanner(new File("C:/Users/thanos/jtmp/test/intIn.txt"));
19         PrintStream ps = new PrintStream("C:/Users/thanos/jtmp/test/intOut.txt");
20         String token;
21         int num = 0, sum = 0, count = 0;
22         double average = 0.0;
```

Επεξεργασία αρχείου ακεραίων (3)


```
24          while (sc.hasNext()) {
25              token = sc.next();
26              if (isInt(token)) {
27                  num = Integer.parseInt(token);
28                  count++;
29                  sum += num;
30              }
31          }
32
33          average = (double) sum / count;
34          ps.printf("Το άθροισμα είναι %d%n", sum);
35          ps.printf(Locale.ENGLISH, "Ο μέσος όρος είναι %.2f", average);
36
37          sc.close();
38          ps.close();
39      }
```

Μέσα σε μια while/do διαβάζουμε ***όσο υπάρχει επόμενο*** *token*
Κάθε φορά που ***διαβάζουμε ένα String, ελέγχουμε αν είναι ακέραιος με την isInt()*** *και αν είναι τότε αυξάνουμε τον μετρητή i κατά 1 και επίσης αθροίζουμε στο sum*

- Κάνουμε ***casting το sum σε double*** και όλη η παράσταση γίνεται double γιατί διαφορετικά το αποτέλεσμα θα είναι int.
- Επίσης, χρησιμοποιούμε ***Locale US ώστε η υποδιαστολή να είναι η τελεία*** διαφορετικά το default Locale ήταν ***el_GR*** και η υποδιαστολή θα ήταν το κόμμα

Μέθοδος isInt(String s)


```
42      /**
43       * Ελέγχει αν το String s είναι αριθμός.
44       *
45       * @param s το προς έλεγχο String
46       * @return true, αν το s είναι αριθμός, αλλιώς false.
47       */
48      public static boolean isInt(String s) {
49          try {
50              Integer.parseInt(s);
51              return true;
52          } catch (NumberFormatException e) {
53              return false;
54          }
55      }
56  }
```

- Ελέγχει αν ένα String είναι ακέραιος

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 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()*** με μία διαφορά, ότι η Scanner καταναλώνει την new line, ενώ η readLine() όχι

Scanner vs BufferedReader

- 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)

BufferedReader


```java
package testbed.ch9;

import java.io.*;

public class BufferedReaderApp {

    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();

        try (BufferedReader bf = new BufferedReader(new FileReader("C:/tmp/f-reader.txt"))) {
            String line = "";

            while ((line = bf.readLine()) != null) {
                sb.append(line).append("\n");
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }

        System.out.println(sb);
    }
}
```

- Ο ***BufferedReader*** λαμβάνει ως παράμετρο ένα *Reader* που εδώ είναι ***FileReader*** γιατί θα διαβάσουμε αρχείο χαρακτήρων

BufferedReader και java.io (1)


```java
package testbed.ch9;

import java.io.*;
import java.nio.charset.StandardCharsets;

public class CitiesIOApp {

    public static void main(String[] args) {
        String line;
        String[] cities;
        File dir = new File("C:/tmp/io");

        if (!dir.exists()) {
            if (!dir.mkdir()) {
                System.err.println("Error in mkdir");
                System.exit(1);
            }
        }
```

- Παράδειγμα χρήσης του ***BufferedReader***, όπου θα διαβάζει γραμμή-γραμμή τα στοιχεία χωρών
- Χρησιμοποιούμε τη βιβλιοθήκη java.io
- Αρχικά δημιουργούμε ένα directory για output με dir.mkdir() ή dir.mkdirs() αν δεν υπάρχει όλο το path

BufferedReader και java.io (2)


```
20      try (BufferedReader bf = new BufferedReader(new FileReader("C:/tmp/cities.txt"))) {
21          File grFile, deFile, usaFile;
22
23          while ((line = bf.readLine()) != null) {
24              cities = line.split(" +");
25              switch (cities[0]) {
26                  case "Greece":
27                      grFile = new File(dir + "/" + "gr.txt");
28                      PrintStream gr = new PrintStream(grFile, StandardCharsets.UTF_8);
29                      print(gr, "GR Cities");
30                      print(gr, cities);
31                      break;
32                  case "USA":
33                      usaFile = new File(dir + "/" +"/usa.txt");
34                      PrintStream usa = new PrintStream(usaFile, StandardCharsets.UTF_8);
35                      print(usa, "USA Cities");
36                      print(usa, cities);
37                      break;
38                  case "Germany":
39                      deFile = new File(dir + "/" + "/de.txt");
40                      PrintStream de = new PrintStream(deFile, StandardCharsets.UTF_8);
41                      print(de, "DE Cities");
42                      print(de, cities);
43                      break;
44                  default:
```

- Διαβάζουμε γραμμή-γραμμή μέχρι να βρούμε τέλος αρχείου, δηλ. null
- Κάθε γραμμή που διαβάζουμε την “σπάμε” με split με βάση ένα ή περισσότερα κενά διαστήματα σε ένα πίνακα από Strings με όνομα ***splitted*** (το + μετά το κενό στο split σημαίνει ένα ή περισσότερα κενά διαστήματα)
- Στη συνέχεια ελέγχουμε το ***splitted[0]*** που είναι το όνομα της χώρας και γράφουμε XXX Cities ενώ μετά ακολουθεί επίσης η ***print***() (υπερφορτωμένη) που εκτυπώνει τα ονόματα των πόλεων

BufferedReader και java.io (3)


```
44                      default:
45                          System.out.println("Error in Cities");
46                  }
47              }
48          } catch (IOException ex) {
49              ex.printStackTrace();
50          }
51      }
52
53  @   public static void print(PrintStream ps, String[] tokens) {
54          for (int i = 1; i < tokens.length; i++) {
55              ps.print(tokens[i] + " ");
56          }
57      }
58
59  @   public static void print(PrintStream ps, String message) {
60          ps.println(message);
61      }
62  }
```

- Η print() είναι υπερφορτωμένη (overloaded) – ίδιο όνομα διαφορετικές παράμετροι-
- Η 1η print() εκτυπώνει ένα πίνακα String[] tokens
- Η 2η print() εκτυπώνει ένα μήνυμα

BufferedReader και java.io (4)


```
23          while ((line = bf.readLine()) != null) {
24              cities = line.split(" +");
```

- Παρατηρούμε στις γραμμές 23 και 24, 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

NIO (New IO)

- interface **Path**
  - Το **Path** αντικαθιστά το **File** της java.io
- **Paths** (utility class)
  - Δημιουργεί (creates) directories και files
  - Συνενώνει Directory paths με file paths
  - Αντιγράφει αρχεία

nio (1)


```
1   package testbed.ch9;
2
3   import java.io.*;
4   import java.nio.charset.StandardCharsets;
5   import java.nio.file.*;
6
7   public class CitiesNIOApp {
8
9       public static void main(String[] args) {
10          String line;
11          String[] cities;
12
13          try (BufferedReader bf = new BufferedReader(new FileReader("C:/tmp/cities.txt"))) {
14              Path dir = Paths.get("C:/tmp/cities");
15              if (Files.notExists(dir)) Files.createDirectory(dir);
16              Path path;
17
```

- Αντί του File χρησιμοποιούμε το Path καθώς και τις utility κλάσεις Paths και Files

nio (2)


```
18          while ((line = bf.readLine()) != null) {
19              cities = line.split(" +");
20
21              switch (cities[0]) {
22                  case "Greece":
23                      // path = Paths.get("C:/tmp/gr.txt");
24                      path = dir.resolve("gr.txt");
25                      PrintStream gr = new PrintStream(path.toFile(), StandardCharsets.UTF_8);
26                      print(gr, "GR Cities");
27                      print(gr, cities);
28                      break;
29                  case "USA":
30                      // path = Paths.get("C:/tmp/usa.txt");
31                      path = dir.resolve("usa.txt");
32                      PrintStream usa = new PrintStream(path.toFile(), StandardCharsets.UTF_8)
33                      print(usa, "USA Cities");
34                      print(usa, cities);
35                      break;
36                  case "Germany":
37                      // path = Paths.get("C:/tmp/de.txt");
38                      path = dir.resolve("de.txt");
39                      PrintStream de = new PrintStream(path.toFile(), StandardCharsets.UTF_8);
40                      print(de, "DE Cities");
41                      print(de, cities);
42                      break;
```

- Η resolve συνενώνει προσθέτοντας στο τέλος του dir

nio (3)


```
43                      default:
44                          System.out.println("Error in Cities");
45                  }
46              }
47          } catch (IOException ex) {
48              ex.printStackTrace();
49          }
50      }
51
52  @   public static void print(PrintStream ps, String[] tokens) {
53          for (int i = 1; i < tokens.length; i++) {
54              ps.print(tokens[i] + " ");
55          }
56      }
57
58  @   public static void print(PrintStream ps, String message) {
59          ps.println(message);
60      }
61  }
```

- Βοηθητικές μέθοδοι, που χρησιμοποιούμε για να εκτυπώσουμε

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***

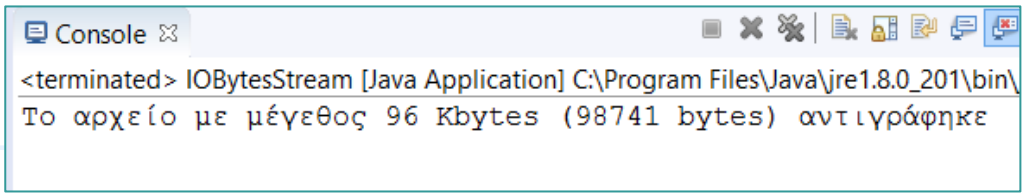
Αντιγραφή εικόνας (1)

- 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;
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 testbed.ch9;

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

/**
 * Αντιγράφει ένα αρχείο βίντεο. Υπολογίζει το χρόνο
 * αντιγραφής. Χρησιμοποιεί FileInputStream και
 * FileOutputStream διαβάζοντας και γράφοντας ένα
 * byte τη φορά.
 *
 * @author A. Androutsos
 */
public class IOVideoCopy1 {

    public static void main(String[] args) throws java.io.IOException {
        int b, count = 0;

        try (FileInputStream in = new FileInputStream("C:/tmp/ArtGalleries-img/Art_Basel.jpg");
             FileOutputStream out = new FileOutputStream("C:/tmp/ArtGalleries-img/art-out.jpg")) {

            long start = System.nanoTime();
            // Αντέγραψε το αρχείο
            while ((b = in.read()) != -1) {
                out.write(b);
                count++;
            }
            long end = System.nanoTime();
            long elapsed = end - start;

            System.out.printf("Το αρχείο με μέγεθος %d KBytes (%d bytes) αντιγράφηκε%n",
                    count / 1024, count);
            System.out.printf("Time: %.2f seconds", elapsed / 1_000_000_000.0);
        }
    }
}
```

- 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

```
Run:  IOVideoCopy1
"C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe" "
Το αρχείο με μέγεθος 3109 KBytes (3184219 bytes) αντιγράφηκε
Time: 18,65 seconds
Process finished with exit code 0
```

Αντιγραφή με buffer


```java
package testbed.ch9;

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

/**
 * Αντιγράφει ένα αρχείο βίντεο. Υπολογίζει το χρόνο
 * αντιγραφής. Χρησιμοποιεί FileInputStream και
 * FileOutputStream διαβάζοντας και γράφοντας 8192 bytes
 * (= 8 KBytes) τη φορά.
 *
 * @author A. Androutsos
 */
public class IOVideoCopy2 {

    public static void main(String[] args) throws java.io.IOException {
        int b, count = 0;
        byte[] buf = new byte[8192];

        try (FileInputStream in = new FileInputStream("C:/tmp/ArtGalleries-img/Art_Basel.jpg");
             FileOutputStream out = new FileOutputStream("C:/tmp/ArtGalleries-img/art-out.jpg")) {

            long start = System.nanoTime();

            // Αντέγραψε το αρχείο
            while ((b = in.read(buf)) != -1) {
                out.write(b);
                count += b;
            }

            long end = System.nanoTime();
            long elapsed = end - start;

            System.out.printf("Το αρχείο με μέγεθος %d KBytes (%d bytes) αντιγράφηκε%n", count / 1024,
                    count);
            System.out.printf("Time: %.2f seconds", elapsed / 1_000_000_000.0);

        }
    }
}
```

- 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***

```
Run:  IOVideoCopy3
"C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe" "-
Το αρχείο με μέγεθος 34599 KBytes (35429453 bytes) αντιγράφηκε
Time: 0,10 seconds
Process finished with exit code 0
```

Buffering με BufferedInputStream


```java
package testbed.ch9;

import java.io.*;

/**
 * Αντιγράφει ένα αρχείο βίντεο. Υπολογίζει το χρόνο
 * αντιγραφής. Χρησιμοποιεί BufferedInputStream, BufferedOutputStream,
 * FileInputStream, FileOutputStream διαβάζοντας και γράφοντας 8192 bytes
 * (= 8 KBytes) τη φορά.
 *
 * @author A. Androutsos
 */
public class IOVideoCopy3 {

    public static void main(String[] args) {
        int b, count = 0;
        byte[] buf = new byte[8192];

        try (BufferedInputStream in = new BufferedInputStream(
                new FileInputStream("C:/tmp/v1.mp4"));
             BufferedOutputStream out = new BufferedOutputStream(
                     new FileOutputStream("C:/tmp/v-out2.mp4")) ) {
            long start = System.nanoTime();

            // Αντέγραψε το αρχείο
            while ((b = in.read(buf, 0, buf.length - 1)) != -1) {
                out.write(buf, 0, b);
                count += b;
            }

            long end = System.nanoTime();
            long elapsed = end - start;

            System.out.printf("Το αρχείο με μέγεθος %d KBytes (%d bytes) αντιγράφηκε%n",
                    count / 1024, count);
            System.out.printf("Time: %.2f seconds", elapsed / 1_000_000_000.0);

        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

- Αυτό που έχει αλλάξει είναι πως αντί για File Stream έχουμε Buffered Stream
- Ο χρόνος είναι παρόμοιος

```
Run:  IOVideoCopy3
"C:\Program Files\Amazon Corretto\jdk11.0.10_9\bin\java.exe" "-
Το αρχείο με μέγεθος 34599 KBytes (35429453 bytes) αντιγράφηκε
Time: 0,10 seconds
Process finished with exit code 0
```

Μικρή Εργασία

- Έστω ένα αρχείο locations.txt με τοποθεσίες, latitude και longitude. Αναπτύξτε ένα πρόγραμμα που να διαβάζει και να γράφει σε ένα άλλο αρχείο με το συγκεκριμένο format

```
Location,Latitude,Longitude
Athens,37.9838,23.7275
Thessaloniki,40.6401,22.9444
Patras,38.2466,21.7346
Heraklion,35.3387,25.1442
```

```
{ location: 'Athens', latitude: 37.9838, longitude: 23.7275 },
{ location: 'Thessaloniki', latitude: 40.6401, longitude: 22.9444 },
{ location: 'Patras', latitude: 38.2466, longitude: 21.7346 },
{ location: 'Heraklion', latitude: 35.3387, longitude: 25.1442 }
```