

ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

String processing

Αρχεία

Δομές

STRING PROCESSING

Strings

- Η επεξεργασία αλφαριθμητικών είναι πολύ σημαντική για πολλές εφαρμογές. Θα δούμε μερικές χρήσιμες εντολές
- Σε όλες τις εντολές για επεξεργασία των Strings δεν πρέπει να ξεχνάμε ότι τα Strings είναι **immutable objects**
 - Οι **μέθοδοι** που καλεί μια μεταβλητή String **δεν μπορούν να αλλάξουν** την μεταβλητή, μόνο να επιστρέψουν ένα **νέο String**.

toLowerCase, trim

- Οι παρακάτω εντολές είναι χρήσιμες για να **κανονικοποιούμε** το String
 - **toLowerCase()**: μετατρέπει όλους τους χαρακτήρες ενός String σε μικρά γράμματα.
 - **trim()**: αφαιρεί λευκούς χαρακτήρες από την αρχή και το τέλος
- Χρήσιμες εντολές όταν κάνουμε **συγκρίσεις** μεταξύ Strings και θέλουμε να τα φέρουμε σε κοινή μορφή.

Παράδειγμα

```
public class StringTest1
{
    public static void main(String args[]) {
        String s1 = "this is a sentence ";
        String s2 = "This is a sentence";

        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s1.equals(s2));

        s1 = s1.trim();
        s2 = s2.toLowerCase();
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s1.equals(s2));
    }
}
```

Για να αποφεύγονται κενά στην αρχή ή στο τέλος

Χρήσιμη εντολή για συγκρίσεις λέξεων, για να μην εξαρτόμαστε αν η λέξη είναι σε μικρά ή κεφαλαία

Πρέπει **πάντα** να γίνεται ξανά ανάθεση στη μεταβλητή. Η εντολή `s2.toLowerCase()`; δεν αλλάζει το s2 επιστρέφει το αλλαγμένο String.

split

- Η εντολή `split` είναι χρήσιμη για να σπάμε ένα `String` σε **πεδία** που διαχωρίζονται από ένα συγκεκριμένο `string`
 - **Όρισμα**: το `string` ως προς το οποίο θέλουμε να σπάσουμε το κείμενο.
 - **Επιστρέφει**: πίνακα `String[]` με τα πεδία που δημιουργήθηκαν.

Παράδειγμα: από το String:

“Student: Bob Marley AM: 111”

θέλουμε το όνομα του φοιτητή και το AM του

```
class SplitTest1{
    public static void main(String args[]){
        String s = "Student: Bob Marley\tAM: 111";
        System.out.println(s);

        String fields[] = s.split("\t");

        String studentFields[] = fields[0].split(":");
        String studentName = studentFields[1].trim();

        String AMFields[] = fields[1].split(":");
        int studentAM = Integer.parseInt(AMFields[1].trim());

        System.out.println(studentName + "\t" + studentAM);
    }
}
```

Split πρώτα ως προς “\t”
και μετά ως προς “:”

Χρήση της trim

replace

- Η εντολή είναι χρήσιμη αν θέλουμε να αλλάξουμε κάπως το String
 - `replace(String before, String after)`: αντικαθιστά το `before` με το `after` και **επιστρέφει** το αλλαγμένο String

Παράδειγμα

```
class ReplaceTest1
{
    public static void main(String[] args){
        String s1 = "Is this a greek question?";
        System.out.println("Before:" + s1);
        s1 = s1.replace("?", ";");
        System.out.println("After:" + s1);

        String s2 = "This is not a question?";
        System.out.println("Before:" + s2);
        s2 = s2.replace("?", "");
        System.out.println("After:" + s2);

        String s3 = "20-5-2013";
        System.out.println("Before:" + s3);
        s3 = s3.replace("-", "/");
        System.out.println("After:" + s3);
    }
}
```

Αντικαθιστά το "?" με ";"

Σβήνει το "?"

Αντικαθιστά όλα τα "-" με "/"

Split και Replace

- Υπάρχουν περιπτώσεις που θέλουμε να σπάσουμε ή να αντικαταστήσουμε με βάση κάτι πιο **περίπλοκο** από ένα String
 - Π.χ., θέλουμε να σπάσουμε ένα String ως προς **tabs** ή **κενά**
 - Π.χ., θέλουμε να σβήσουμε οτιδήποτε είναι **ερωτηματικό, ελληνικό** ή **αγγλικό**
 - Π.χ., θέλουμε να σβήσουμε τις τελείες αλλά **μόνο** αν είναι **στο τέλος του String**.
- Για να προσδιορίσουμε τέτοιες περίπλοκες περιπτώσεις χρησιμοποιούμε **κανονικές εκφράσεις (regular expressions)**

Regular Expressions

- Ένας τρόπος να περιγράψουμε Strings που έχουν ακολουθούν ένα **κοινό μοτίβο**
 - Έχετε ήδη χρησιμοποιήσει κανονικές εκφράσεις. Όταν γράφετε “ls *.txt” το “*.txt” είναι μια κανονική έκφραση που περιγράφει όλα τα Strings που τελειώνουν σε “.txt”

Κανονικές Εκφράσεις στη Java

- Μπορείτε να διαβάσετε μια περίληψη [στη σελίδα της Oracle](#)
- Οι κανονικές εκφράσεις μπορούν να περιγράψουν πολλά πράγματα. Εμείς θα χρησιμοποιήσουμε κάποιες απλές εκφράσεις.
- Παραδείγματα:
 - `[abc]`: a ή b ή c
 - `^a` : Ξεκινάει με a
 - `a$`: τελειώνει με a
 - `\s` ή `\p{Space}`: white space (κενό, tab, αλλαγή γραμμής)
 - `\p{Punct}`: όλα τα σημεία στίξης
- Για να **χρησιμοποιήσουμε** τις κανονικές εκφράσεις τις μετατρέπουμε σε ένα **string** που δίνεται ως όρισμα στην `split` ή την `replaceAll`.
 - Π.χ. `"[abc]"`, `"^a"`, `"a$"`, `"\s"`, `"\p{Space}"`, `"\p{Punct}"`
 - Χρειαζόμαστε το `"\"` ώστε να βάλουμε το `\` μέσα στο string.

Παράδειγμα

```
class SplitTest2
{
    public static void main(String args[]){
        String s1 = "sentence 1\tsentence 2";
        String[] tokens = s1.split("[\t ]");
        for (String t: tokens){
            System.out.println(t);
        }
        tokens = s1.split("\\s");
        for (String t: tokens){
            System.out.println(t);
        }

        String s2 = "To be or not to be? This is the question. The
question we must face";
        String[] sentences = s2.split("[?.]");
        for (String s: sentences){
            System.out.println(s.trim());
        }
    }
}
```

Split στο tab και το κενό

Split σε οποιοδήποτε
white space

Split στο ερωτηματικό
και την τελεία

Παράδειγμα

Για να χρησιμοποιήσουμε την κανονική έκφραση χρειαζόμαστε την εντολή `replaceAll`

```
class ReplaceTest2
{
    public static void main(String args[]) {
        String s = "The cost is 99.99 dollars.";
        System.out.println(s);
        s = s.replaceAll("[.]$", "");
        System.out.println(s);

        s = "\"Quoted (\"quote\") text\"";
        System.out.println(s);
        s = s.replaceAll("^\"", "");
        s = s.replaceAll("\"$", "");
        System.out.println(s);

        s = "What?Yes!No...";
        System.out.println(s);
        s = s.replaceAll("[.!?]", " ");
        System.out.println(s);
    }
}
```

Σβήνει την τελεία στο **τέλος** του String

Σβήνει το “ στην **αρχή** του String

Σβήνει το ” στο **τέλος** του String

Αντικαθιστά τελεία, θαυμαστικό και ερωτηματικό με κενό.

StringTokenizer

- Η διαδικασία του να σπάμε ένα string σε κομμάτια που χωρίζονται με κενά λέγεται **tokenization** και τα κομμάτια **tokens**.
- Η κλάση [StringTokenizer](#) κάνει και το tokenization και μας επιτρέπει να διατρέχουμε τα tokens
 - `nextToken()`: επιστρέφει το επόμενο token
 - `hasMoreTokens()`: μας λέει αν έχουμε άλλα tokens
- Θα μπορούσαμε να χρησιμοποιήσουμε και την **split** αλλά η **StringTokenizer** χειρίζεται **αυτόματα** τις διάφορες περιπτώσεις με white space
 - Π.χ. πολλαπλά κενά

Παράδειγμα

```
import java.util.StringTokenizer;

class StringTokenizerTest
{
    public static void main(String args[]){
        String s = "Line with tab\t and space";
        System.out.println(s);

        System.out.println("Split tokenization");
        String[] tokens1 = s.split("\\s");
        for (String t: tokens1){
            System.out.println("-"+t+"-");
        }

        System.out.println("StringTokenizer tokenization");
        StringTokenizer tokens2 = new StringTokenizer(s);
        while (tokens2.hasMoreTokens()){
            System.out.println("-"+tokens2.nextToken()+"-");
        }
    }
}
```

Split σε κενό και tab

Δημιουργεί κενό token όταν βρει το "\t "

Δεν δημιουργεί κενό token όταν βρει το "\t "

StringBuilder

- Τα Strings είναι **immutable objects**. Αυτό σημαίνει ότι για να αλλάξουμε ένα String πρέπει να το **ξανα-δημιουργήσουμε** και να το **αντιγράψουμε**
- Για τέτοιου είδους αλλαγές είναι καλύτερα να χρησιμοποιούμε το **StringBuilder**
 - **append(String)**: προσθέτει ένα String στο τέλος
 - **toString()**: επιστρέφει το τελικό String
- Πολύ βολικό για να δημιουργούμε String **συνενώνοντας** πολλαπλά Strings.

```
import java.lang.StringBuilder;
```

Θέλουμε να δημιουργήσουμε ένα String με τους αριθμούς από το 1 ως το N

```
class StringBuilderTest
```

```
{  
    public static void main(String[] args){  
        int N = 100000;
```

```
        String s = "";  
        for (int i = 0; i < 100000; i ++){  
            s = s + " " + i;  
        }  
        System.out.println(s);
```

```
        StringBuilder sb = new StringBuilder("");  
        for (int i = 0; i < 100000; i ++){  
            sb.append(" " + i);  
        }  
        System.out.println(sb.toString());
```

```
    }
```

```
}
```

Ο μπλε κώδικας είναι **πολύ** πιο γρήγορος από τον πράσινο
Ο πράσινος αντιγράφει το String N φορές

APXEIA

Ρεύματα

- Τι είναι ένα **ρεύμα** (ροή)? Μια **αφαίρεση** που αναπαριστά μια **ροή δεδομένων**
 - Η ροή αυτή μπορεί να είναι **εισερχόμενη** προς το πρόγραμμα (μια **πηγή** δεδομένων) οπότε έχουμε **ρεύμα εισόδου**.
 - Παράδειγμα: το πληκτρολόγιο, ένα αρχείο που ανοίγουμε για διάβασμα
 - Ή μπορεί να είναι **εξερχόμενη** από το πρόγραμμα (ένας **προορισμός** για τα δεδομένα) οπότε έχουμε ένα **ρεύμα εξόδου**.
 - Παράδειγμα: Η οθόνη, ένα αρχείο που ανοίγουμε για διάβασμα.
- Όταν δημιουργούμε το ρεύμα το **συνδέουμε** με την ανάλογη πηγή, ή προορισμό.

Βασικά ρεύματα εισόδου/εξόδου

- Ένα **ρεύμα** είναι ένα **αντικείμενο**. Τα βασικά ρεύματα εισόδου/εξόδου είναι έτοιμα αντικείμενα τα οποία ορίζονται σαν πεδία (**στατικά**) της κλάσης **System**
- **System.out**: Το **βασικό ρεύμα εξόδου** που αναπαριστά την οθόνη.
 - Έχει στατικές μεθόδους με τις οποίες μπορούμε να τυπώσουμε στην οθόνη.
- **System.in**: Το **βασικό ρεύμα εξόδου** που αναπαριστά την οθόνη.
 - Χρησιμοποιούμε την κλάση **Scanner** για να πάρουμε δεδομένα από το ρεύμα.
- Μια εντολή εισόδου/εξόδου έχει αποτέλεσμα το λειτουργικό να πάρει ή να στείλει δεδομένα από/προς την αντίστοιχη πηγή/προορισμό.
- Ένα επιπλέον ρεύμα: **System.err**: Ρεύμα για την εκτύπωση **λαθών** στην οθόνη
 - Μας επιτρέπει την ανακατεύθυνση της εξόδου.

Παράδειγμα

```
class SystemErrTest
{
    public static void main(String args[]) {
        System.err.println("Starting program");
        for (int i = 0; i < 10; i ++){
            System.out.println(i);
        }
        System.err.println("End of program");
    }
}
```

Και τα δύο τυπώνουν στην οθόνη αλλά αν κάνουμε ανακατεύθυνση μόνο το System.out ανακατευθύνεται

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

```
public class TextFileOutputDemo2  
{  
    public static void main(String[] args)  
    {  
        PrintWriter outputWriter = null;  
        try  
        {  
            outputWriter = new PrintWriter(new FileOutputStream("stuff.txt"));  
        }  
  
        catch (FileNotFoundException e)  
        {  
            System.out.println("Error opening the file stuff.txt.");  
            System.exit(0);  
        }  
  
        System.out.println("Writing to file.");  
  
        outputWriter.println("The quick brown fox");  
        outputWriter.println("jumped over the lazy dog.");  
  
        outputWriter.close( );  
  
        System.out.println("End of program.");  
    }  
}
```

Το αντικείμενο FileOutputStream δημιουργεί το ρεύμα εξόδου

Το αντικείμενο PrintWriter γράφει στο ρεύμα εξόδου

Η δημιουργία του ρεύματος σε try-catch block

Προσάρτηση σε αρχείο

- Τι γίνεται αν θέλουμε να προσθέσουμε (append) επιπλέον δεδομένα σε ένα υπάρχον αρχείο
 - Ο constructor της `FileOutputStream` που ξέρουμε θα σβήσει τα περιεχόμενα και θα το ξαναγράψουμε από την αρχή.
- Γι αυτό το σκοπό χρησιμοποιούμε ένα άλλο constructor

```
FileOutputStream outputStream =  
    new FileOutputStream("stuff.txt", true);
```

- Το όρισμα `true` υποδηλώνει ότι θέλουμε να προσθέσουμε (append) στο αρχείο
- Αν `δεν υπάρχει` το αρχείο τότε δημιουργείται κανονικά, κενό όπως και πριν.


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

```
public class TextFileOutputDemo2
```

```
{  
    public static void main(String[] args)  
    {  
        PrintWriter outputWriter = null;  
        try  
        {  
            outputWriter = new PrintWriter(new FileOutputStream("stuff.txt", true));  
        }  
  
        catch (FileNotFoundException e)  
        {  
            System.out.println("Error opening the file stuff.txt.");  
            System.exit(0);  
        }  
  
        System.out.println("Writing to file.");  
  
        outputWriter.println("The quick brown fox");  
        outputWriter.println("jumped over the lazy dog.");  
  
        outputWriter.close( );  
  
        System.out.println("End of program.");  
    }  
}
```

Το αντικείμενο FileOutputStream δημιουργεί το ρεύμα εξόδου

Το αντικείμενο PrintWriter γράφει στο ρεύμα εξόδου

Η δημιουργία του ρεύματος σε try-catch block

```
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
```

```
public class TextFileScannerDemo
{
    public static void main(String[] args)
    {
```

```
        Scanner inputReader = null;
```

```
        try
```

```
        {
```

```
            inputReader =
```

```
                new Scanner(new FileInputStream("morestuff.txt"));
```

```
        }
```

```
        catch(FileNotFoundException e)
```

```
        {
```

```
            System.out.println("File morestuff.txt was not found");
```

```
            System.out.println("or could not be opened.");
```

```
            System.exit(0);
```

```
        }
```

```
        String line = inputReader.nextLine( );
```

```
        System.out.println("The line read from the file is:");
```

```
        System.out.println(line);
```

```
        inputStream.close( );
```

```
    }
```

```
}
```

Το αντικείμενο Scanner μας επιτρέπει να διαβάσουμε από το ρεύμα εισόδου

Το αντικείμενο FileInputStream είναι το ρεύμα εισόδου

Η δημιουργία του ρεύματος σε try-catch block

Διαβάζουμε από ένα αρχείο και γράφουμε τις γραμμές με νούμερα.

```
import java.util.Scanner;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.PrintWriter;  
import java.io.FileOutputStream;
```

```
public class ReadWriteDemo  
{  
    public static void main(String[] args)  
    {  
        Scanner inputStream = null;  
        PrintWriter outputStream = null;  
  
        try  
        {  
            inputStream = new Scanner(new FileInputStream("original.txt"));  
            outputStream = new PrintWriter(new FileOutputStream("numbered.txt"));  
        }  
        catch(FileNotFoundException e)  
        {  
            System.out.println("Problem opening files.");  
            System.exit(0);  
        }  
  
        String line = null; int count = 0;  
  
        while (inputStream.hasNextLine( ))  
        {  
            line = inputStream.nextLine( );  
            count++;  
            outputStream.println(count + " " + line);  
        }  
  
        inputStream.close( );  
        outputStream.close( );  
    }  
}
```

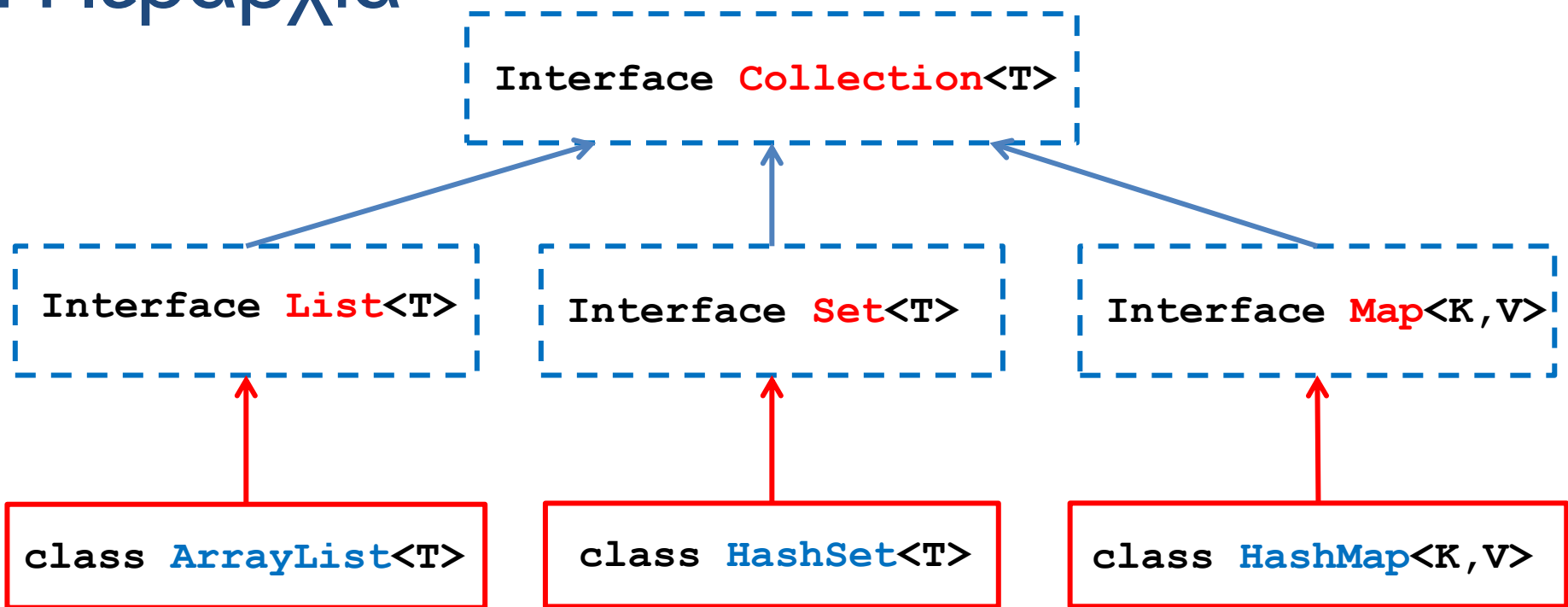
Χρήση των εξαιρέσεων για έλεγχο

```
import java.util.Scanner;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.PrintWriter;  
import java.io.FileOutputStream;
```

```
public class ReadWriteDemo  
{  
    public static void main(String[] args)  
    {  
        Scanner keyboard = new Scanner(System.in);  
        String inputFilename = keyboard.next();  
        String outputFilename = keyboard.next();  
  
        Scanner inputStream = null;  
        PrintWriter outputStream = null;  
  
        boolean openedFilesOk = false;  
        while (!openedFilesOk)  
        {  
            try  
            {  
                inputStream = new Scanner(new FileInputStream(inputFilename));  
                outputStream = new PrintWriter(new FileOutputStream(outputFilename));  
                openedFilesOk = true;  
            }  
            catch (FileNotFoundException e)  
            {  
                System.out.println("Problem opening files. Enter names again:");  
                inputFilename = keyboard.next();  
                outputFilename = keyboard.next();  
            }  
        }  
  
        <υπόλοιπος κώδικας...>  
    }  
}
```

ΔΟΜΕΣ

Η ιεραρχία



Αποθηκεύει δεδομένα σε **σειριακή** μορφή. Υπάρχει η έννοια της **διάταξης**. Καλό αν θέλουμε να **διατρέχουμε** τα δεδομένα συχνά και γρήγορα.

Αποθηκεύει δεδομένα σαν **σύνολο** χωρίς διάταξη. Καλό αν θέλουμε να βρίσκουμε γρήγορα αν ένα στοιχείο **ανήκει** στο σύνολο

Αποθηκεύει **(key,value)** ζεύγη. Παρόμοια δομή με το `HashSet` για την αποθήκευση των **κλειδιών**, αλλά τώρα κάθε κλειδί (key) **σχετίζεται** με μία **τιμή** (value).

ArrayList ([JavaDocs link](#))

- Constructors

- `ArrayList<T> myList = new ArrayList<T>();`
- `ArrayList<T> myList = new ArrayList<T>(10);`
//λίστα με χωρητικότητα 10

- Μέθοδοι

- `add(T x)` : προσθέτει το στοιχείο `x` στο τέλος του πίνακα.
- `add(int i, T x)` : προσθέτει το στοιχείο `x` στη θέση `i` και μετατοπίζει τα υπόλοιπα στοιχεία κατά μια θέση.
- `remove(int i)` : αφαιρεί το στοιχείο στη θέση `i`
- `set(int i, T x)` : αλλάζει την τιμή της θέσης `i` με την τιμή `x`
- `get(int i)` : επιστρέφει την τιμή στη θέση `i`.
- `size()` : ο αριθμός των στοιχείων του πίνακα.

- Διατρέχοντας τον πίνακα:

- `ArrayList<T> myList = new ArrayList<T>();`
- `for(T x: myList) {...}`

HashSet ([JavaDocs link](#))

- Constructors

- `HashSet<T> mySet = new HashSet<T>();`

- Μέθοδοι

- `add(T x)` : προσθέτει το στοιχείο `x` αν δεν υπάρχει ήδη στο σύνολο.

- `remove(T x)` : αφαιρεί το στοιχείο `x`.

- `contains(T x)` : boolean αν το σύνολο περιέχει το στοιχείο `x` ή όχι.

- `size()` : ο αριθμός των στοιχείων στο σύνολο.

- `isEmpty()` : boolean αν έχει στοιχεία το σύνολο ή όχι.

- `Object[] toArray()` : επιστρέφει πίνακα με τα στοιχεία του συνόλου (επιστρέφει πίνακα από Objects – χρειάζεται casting μετά).

- Διατρέχοντας τα στοιχεία του συνόλου:

- `HashSet<T> mySet = new HashSet<T>();`

- `for (T x: mySet) {...}`

HashMap ([JavaDocs link](#))

- Constructors

- `HashMap<K,V> myMap = new HashMap<K,V> ();`

- Μέθοδοι

- `put(K key, V value)` : προσθέτει το ζευγάρι (`key, value`) (δημιουργεί μία συσχέτιση)
 - `V get(K key)` : επιστρέφει την τιμή για το κλειδί `key`.
 - `remove(K key)` : αφαιρεί το ζευγάρι με κλειδί `key`.
 - `containsKey(K key)` : `boolean` αν το σύνολο περιέχει το κλειδί `key` ή όχι.
 - `containsValue(V value)` : `boolean` αν το σύνολο περιέχει την τιμή `value` ή όχι. (αργό)
 - `size()` : ο αριθμός των στοιχείων (ζεύγη από κλειδιά-τιμές) στο `map`.
 - `isEmpty()` : `boolean` αν έχει στοιχεία το `map` ή όχι.
 - `Set<K> keySet()` : επιστρέφει ένα `Set` με τα κλειδιά.
 - `Collection<V> values()` : επιστρέφει ένα `Collection` με τις τιμές

Παράδειγμα

- Έχουμε ένα αρχείο `studentNames.txt` με τα ΑΜ και τα ονόματα των φοιτητών (tab-separated) και ένα αρχείο `studentGrades.txt` με τα ΑΜ και βαθμό (για κάποια μαθήματα – ένα μάθημα ανά γραμμή). Τυπώστε σε ένα αρχείο ΑΜ, όνομα, βαθμό.

```
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.io.FileOutputStream;

import java.util.HashMap;

class Join
{
    public static void main(String[] args){
        Scanner nameInputStream = null;
        Scanner gradesInputStream = null;
        PrintWriter outputStream = null;

        try
        {
            nameInputStream = new Scanner(
                new FileInputStream("studentNames.txt"));
            gradesInputStream = new Scanner(
                new FileInputStream("studentGrades.txt"));
            outputStream = new PrintWriter(
                new FileOutputStream("studentNamesGrades.txt"));
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Problem opening files.");
            System.exit(0);
        }
    }
}
```

Άνοιγμα των αρχείων εισόδου
για διάβασμα και του αρχείου
εξόδου για γράψιμο

Συνέχεια στην
επόμενη

Συνέχεια από
την προηγούμενη

```
String line = null;
```

```
HashMap<Integer,String> namesHash = new HashMap<Integer,String>();
```

```
while (nameInputStream.hasNextLine() )
```

```
{
```

```
    line = nameInputStream.nextLine();
```

```
    String[] fields = line.split("\t");
```

```
    Integer AM = Integer.parseInt(fields[0]);
```

```
    String name = fields[1];
```

```
    namesHash.put(AM,name);
```

```
}
```

```
nameInputStream.close();
```

```
while (gradesInputStream.hasNextLine() )
```

```
{
```

```
    line = gradesInputStream.nextLine();
```

```
    String[] fields = line.split("\t");
```

```
    Integer AM = Integer.parseInt(fields[0]);
```

```
    String grade = fields[1];
```

```
    if (!namesHash.containsKey(AM)) { continue; }
```

```
    String name = namesHash.get(AM);
```

```
    outputStream.println(AM+"\t"+name+"\t"+grade);
```

```
}
```

```
gradesInputStream.close();
```

```
outputStream.close();
```

```
}
```

```
}
```

Διάβασε τα ζεύγη AM, όνομα και βάλε τα σε ένα HashMap με κλειδί το AM

Υποθέτουμε ότι το κάθε AM εμφανίζεται μόνο μία φορά

Διάβασε τα ζεύγη AM, βαθμός και έλεγξε αν το AM εμφανίζεται ως κλειδί στο HashMap.

Αν ναι τύπωσε AM, όνομα και βαθμό στο αρχείο εξόδου