ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αρχεία

Επεξεργασία αλφαριθμητικών

ΑΡΧΕΙΑ

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 γράψιμο.
  - 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 εισόδου που αναπαριστά το πληκτρολόγιο.
  - 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

Αρχεία

- Ένα ρεύμα εξόδου ή εισόδου μπορεί να συνδέεται με ένα αρχείο στο οποίο γράφουμε ή από το οποίο διαβάζουμε.
  - Δύο τύποι αρχείων: Αρχεία κειμένου (ή αρχεία ASCII) και δυαδικά (binary) αρχεία
- Στα αρχεία κειμένου η πληροφορία είναι κωδικοποιημένη σε χαρακτήρες ASCII
  - Πλεονέκτημα: μπορεί να διαβαστεί και από ανθρώπους
- Στα binary αρχεία έχουμε διαφορετική κωδικοποίηση
  - Πλεονέκτημα: πιο γρήγορη η μεταφορά των δεδομένων.
- Εμείς θα ασχοληθούμε με αρχεία κειμένου

Ρεύμα εξόδου σε αρχεία

- Για να γράψουμε σε ένα αρχείο θα πρέπει καταρχάς να δημιουργήσουμε ένα ρεύμα εξόδου που θα συνδέεται με το αρχείο.
- Η Java μας παρέχει την κλάση FileOutputStream η οποία μας επιτρέπει να δημιουργήσουμε ένα τέτοιο ρεύμα.
- Δημιουργία του ρεύματος:

```
FileOutputStream outputStream =
      new FileOutputStream(<ονομα αρχείου>);
```

Παράδειγμα

- `FileOutputStream outputStream =`
  `new FileOutputStream("stuff.txt");`
- Δημιουργεί το αντικείμενο **`outputStream`** το οποίο είναι ένα ρεύμα εξόδου προς το αρχείο με το όνομα **`stuff.txt`**
  - Αν το αρχείο δεν υπάρχει τότε θα δημιουργηθεί ένα κενό αρχείο στο οποίο μπορούμε να γράψουμε
  - Αν υπάρχει ήδη τότε τα περιεχόμενα του θα σβηστούν και γράφουμε και πάλι σε ένα κενό αρχείο

FileNotFoundException

- Η δημιουργία του ρεύματος πετάει μια εξαίρεση FileNotFoundException την οποία πρέπει να πιάσουμε
  - Η δημιουργία του ρεύματος είναι πάντα μέσα σε ένα try-catch block

```
try
{
    FileOutputStream outputStream =
          new FileOutputStream(“stuff.txt”);
}
catch (FileNotFoundException e)
{
    System.out.println("Error opening the file stuff.txt.");
    System.exit(0);
}
```

FileNotFoundException

- Τι σημαίνει FileNotFoundException όταν δημιουργούμε ένα αρχείο?
  - Μπορεί να έχουμε δώσει λάθος path
  - Μπορεί να μην υπάρχει χώρος στο δίσκο
  - Μπορεί να μην έχουμε write access
  - κλπ

Εγγραφή σε αρχείο

- Με την προηγούμενη εντολή συνδέσαμε ένα ρεύμα εξόδου με ένα αρχείο στο δίσκο, στο οποίο θα γράψουμε
- Για να γίνει η εγγραφή πρέπει:
  - Να δημιουργήσουμε ένα αντικείμενο που μπορεί να γράφει στο αρχείο («Ανοίγουμε το αρχείο»)
  - Να καλέσουμε μεθόδους που γράφουν στο αρχείο («Εγγραφή»)
  - Όταν τελειώσουμε να αποδεσμεύσουμε το αντικείμενο από το ρεύμα («Κλείνουμε το αρχείο»)
- Μπορούμε να τα κάνουμε αυτά με την κλάση PrintWriter

PrintWriter

- Constructor:
  - `PrintWriter(FileOutputStream o)`: Παίρνει σαν όρισμα ένα αντικείμενο τύπου FileOutputStream
  - Όταν δημιουργούμε ένα αντικείμενο PrintWriter ανοίγουμε το αρχείο για διάβασμα.
  - Παράδειγμα:
    - `PrintWriter outputWriter = new PrintWriter(outputStream);`
- Μέθοδοι:
  - `print(String s)`: παρόμοια με την print που ξέρουμε αλλά γράφει πλέον στο αρχείο
  - `println(String s)`: παρόμοια με την println που ξέρουμε αλλά γράφει πλέον στο αρχείο
  - `close()`: ολοκληρώνει την εγγραφή (γράφει ότι υπάρχει στο buffer) και κλείνει το αρχείο
  - `flush()`: γράφει ότι υπάρχει στο buffer

Ένα ολοκληρωμένο παράδειγμα

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

public class TextFileOutputDemo1
{
    public static void main(String[] args)
    {
        FileOutputStream outputStream = null;
        try
        {
            outputStream = new FileOutputStream("stuff.txt");
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Error opening the file stuff.txt.");
            System.exit(0);
        }

        PrintWriter outputWriter = new PrintWriter(outputStream);

        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.");
    }
}
```

Πιο συνοπτικός κώδικας

```
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 έτσι κι αλλιώς δεν το χρησιμοποιούμε αλλού. Δημιουργούμε ένα ανώνυμο αντικείμενο.

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

Διάβασμα από αρχείο κειμένου

- Η διαδικασία είναι παρόμοια και για διάβασμα
- Πρώτα δημιουργούμε ένα αντικείμενο τύπου FileInputStream το οποίο συνδέει ένα ρεύμα εισόδου με το όνομα του αρχείου

```
FileInputStream inputStream =
        new FileInputStream(<όνομα αρχείου>);
```

- Μετά θα χρησιμοποιήσουμε την γνωστή μας κλάση Scanner για να:
  - Να ανοίξουμε το αρχείο
    - `Scanner inputReader = new Scanner(inputStream);`
  - Να διαβάσουμε από το αρχείο
    - `inputReader.nextLine();`
  - Να κλείσουμε το αρχείο
    - `inputReader.close();`

Το System.in που χρησιμοποιούσαμε μέχρι τώρα είναι ένα ρεύμα εισόδου

Ένα παράδειγμα

```
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( );
    }
}
```

Η συνοπτική εκδοχή του κώδικα

Scanner

- Η Scanner έχει διάφορες μεθόδους για να διαβάζουμε:
  - nextLine(): διαβάζει μέχρι το τέλος της γραμμής
  - nextInt(): διαβάζει ένα ακέραιο
  - nextDouble(): διαβάζει ένα πραγματικό
  - next(): διαβάζει το επόμενο λεκτικό στοιχείο (χωρισμένο με κενό)
- Έλεγχοι για τέλος εισόδου
  - hasNextLine(): επιστρέφει true αν υπάρχει κι άλλη γραμμή να διαβάσει
  - hasNext(): επιστρέφει true αν υπάρχει κι άλλο String να διαβάσει
  - hasNextInt(): επιστρέφει true αν υπάρχει κι άλλος ακέραιος

Ένα παράδειγμα με διάβασμα και γράψιμο

Διαβάζουμε από ένα αρχείο και γράφουμε τις γραμμές του αριθμημένες σε ένα νέο αρχείο.

```
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);
        }

        int count = 0;
        while (inputStream.hasNextLine( )){
            String line = inputStream.nextLine( );
            count++;
            outputStream.println(count + " " + line);
        }
        inputStream.close( );
        outputStream.close( );
    }
}
```

Η hasNextLine θα επιστρέψει false όταν φτάσουμε στο τέλος του αρχείου

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(outputFilname));
                openedFilesOk = true;
            }
            catch(FileNotFoundException e)
            {
                System.out.println("Problem opening files. Enter names again:");
                inputFilename = keyboard.next();
                outputFilename = keyboard.next();
            }
         }

           <υπόλοιπος κώδικας…>

    }
}
```

Η κλάση File

- Η κλάση File μας δίνει πληροφορίες για ένα αρχείο που θα μπορούσαμε να πάρουμε από το λειτουργικό σύστημα
- Constructor:
  - File fileObject = new File(<όνομα>);
  - Το όνομα συνήθως θα είναι ένα όνομα αρχείου, αλλά μπορεί να είναι και directory.
- Μέθοδοι:
  - exists(): επιστρέφει boolean αν υπάρχει ή όχι το αρχείο/path
  - getName(): επιστρέφει το όνομα του αρχείου από το full path name
  - getPath(): επιστρέφει το path μέχρι το αρχείο από το full path name
  - isFile(): boolean που μας λέει αν το όνομα είναι αρχείο η όχι
  - isDirectory(): boolean που μας λέει αν το όνομα είναι directory η όχι
  - mkdir(): δημιουργεί το directory στο path που δώσαμε ως όρισμα.

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 (κενά, tabs, αλλαγή γραμής) 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 η 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 ΑΜ 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 περιγράφουμε 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”
- Μια κανονική έκφραση λέμε ότι ταιριάζει (matches) με ένα string όταν το string περιγράφεται από το γενικό μοτίβο της κανονικής έκφρασης.

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]: ταιριάζει με a ή b ή c
  - ^a : ταιριάζει με ένα a που εμφανίζεται στην αρχή του String.
  - a$: ταιριάζει με ένα a που εμφανίζεται στο τέλος του String
  - \s ή \p{Space}: ταιριάζει με οποιοδήποτε white space (κενό, tab, αλλαγή γραμμής)
  - \p{Punct}: ταιριάζει όλα τα σημεία στίξης
  - a*: ταιριάζει 0 ή παραπάνω εμφανίσεις του a
  - a+: ταιριάζει 1 ή παραπάνω εμφανίσεις του a

- Για να χρησιμοποιήσουμε τις κανονικές εκφράσεις τις μετατρέπουμε σε ένα string που δίνεται ως όρισμα στην split η την replaceAll.
  - Π.χ. `"[abc]"`, `"^a"`, `"a$"`, `"\\s"`, `"\\p{Space}"`, `"\\p{Punct}"`
    - Χρειαζόμαστε το "\\" ώστε να βάλουμε το \ μέσα στο string.

Παρένθεση

- Ο χαρακτήρας **** λέγεται escape character
  - Όταν τον συνδυάζουμε με άλλους χαρακτήρες παίρνει διαφορετικό νόημα όταν είμαστε μέσα σε String
    - **\n**: αλλαγή γραμμής
    - **\t**: tab
    - **\“**: ο χαρακτήρας **“**
    - ****: ο χαρακτήρας ****

Παράδειγμα

```
class SplitTest2
{
    public static void main(String args[]){
        String s1 = "sentense 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("[.!?]"," ");
        //s = s.replaceAll("\\p{Punct}"," "); // εναλλακτικά
        System.out.println(s);

        s = "Space: Tab:\t:End";
        System.out.println(s);
        s = s.replaceAll("\\p{Space}","");
        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.

Εναλλακτικός τρόπος να αντικαταστήσουμε τα σημεία στίξεως με κενά.

Σβήνει τους whitespace χαρακτήρες

```
class ReplaceTest3
{
    public static void main(String args[]){
        String s = "Hello...";
        s = s.replaceAll("[.]$","");
        System.out.println(s);

        s = s.replaceAll("[.]*$","");
        System.out.println(s);

        s = "\" Quoted (\"quote\") text \"";
        String[] words = s
                .toLowerCase()
                .replaceAll("^\"","")
                .replaceAll("\"$","")
                .trim()
                .split();
        System.out.println(s);
    }
}
```

Τι θα τυπώσει?

Σβήνει μία τελεία από το τέλος του String

Πως μπορούμε να σβήσουμε όλες τις τελείες?

Θέλουμε από το s να αφαιρέσουμε τα αρχικά και τελικά " να αφαιρέσουμε αρχικά και τελικά κενά να μετατρέψουμε τα γράμματα σε μικρά και να το σπάσουμε σε λέξεις

Για να μην κάνουμε συνεχείς αναθέσεις των αποτελεσμάτων των μεθόδων βολεύει να κάνουμε αλυσιδωτές κλήσεις των μεθόδων.

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
  - StringTokenizer st = new StringTokenizer(s): Δημιουργέι ένα tokenizer για το String s, με διαχωριστικό (delimeter) τους λευκούς χαρακτήρες (\s)
  - 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 "

StringTokenizer

- Μπρούμε να κάνουμε tokenization και με διαφορετικά διαχωριστικά. Αυτά τα προσδιορίζουμε στον constructor.
  - `StringTokenizer st =`

    `new StringTokenizer(s,".?!");`
    - Δημιουργέι ένα tokenizer για το String s, με διαχωριστικό (delimeter) την τελεία, το ερωτηματικό και το θαυμαστικό.

```
import java.util.StringTokenizer;

class StringTokenizerTest2
{
   public static void main(String args[]){
       String s = "The first sentence. The second! Third? And,
finally, the last one.";
       System.out.println(s);
       StringTokenizer tokens  = new StringTokenizer(s,".?!");
       System.out.println("Tokenization:");
       while (tokens.hasMoreTokens()){
          System.out.println(tokens.nextToken().trim());
       }
   }
}
```

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: προσθέτει ένα String στο τέλος του υπάρχοντος. Παίρνει σαν όρισμα String ή οποιοδήποτε πρωταρχικό τύπο. Αν πάρει όρισμα κάποιο αντικείμενο καλείται αυτόματα η μέθοδος toString του αντικειμένου.
  - toString(): επιστρέφει το τελικό String

- Πολύ βολικό για να δημιουργούμε String συνενώνοντας πολλαπλά Strings.

```
import java.lang.StringBuilder;

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());
        }
}
```

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

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

```
import java.lang.StringBuilder;

class StringBuilderTest2
{
	public static void main(String[] args){
		StringBuilder sb = new StringBuilder();
		for (int i = 0; i < 10; i ++){
			Person p = new Person("Some Person",i);
			sb.append(p+"\n");
		}
		String s = sb.toString();
		System.out.println(s);
	}
}
```

Καλείται η μέθοδος toString της Person και συνενώνεται στο τέλος του υπάρχοντος String

ΠΑΡΑΔΕΙΓΜΑ

Αρχεία – Επεξεργασία αλφαριθμητικών - Δομές

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

```
        HashMap<Integer,String> namesHash = new HashMap<Integer,String>();
        while (nameInputStream.hasNextLine( ))
        {
           String 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( ))
        {
           String line = gradesInputStream.nextLine( );
           String[] fields = line.split("\t");
           Integer AM = Integer.parseInt(fields[0]);
           String grade = fields[1];
           if (!nameHash.containsKey(AM)){ continue;}
           String name = namesHash.get(AM);
           outputStream.println(AM+"\t"+name+"\t"+grade);
        }

        gradesInputStream.close();
        outputStream.close( );
    }
}
```

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

Υποθέτουμε ότι το κάθε ΑΜ εμφανίζεται μόνο μία φορά

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

Αν ναι τύπωσε ΑΜ, όνομα και βαθμό στο αρχείο εξόδου