ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Εξαιρέσεις

Αρχεία

String processing

ΕΞΑΙΡΕΣΕΙΣ

Εξαιρέσεις

- οι Εξαιρέσεις (Exceptions) είναι ένας μηχανισμός που έχει η Java για να αντιμετωπίζει περιπτώσεις που το πρόγραμμα δεν εξελίσσεται όπως το είχαμε προβλέψει
  - Π.χ., κάνουμε μια διαίρεση και ο παρανομαστής είναι μηδέν
  - Θέλουμε να διαβάσουμε ένα ακέραιο, αλλά η είσοδος είναι ένα String
  - Θέλουμε να διαβάσουμε από ένα αρχείο αλλά δώσαμε λάθος το όνομα.

Μηχανισμός try-throw-catch

- Ο κώδικας που μπορεί να δημιουργήσει εξαίρεση μπαίνει σε ένα try-block
- Αν η εξέλιξη του κώδικα είναι προβληματική εκτελείται η εντολή throw η οποία «πετάει» την εξαίρεση.
- Το πέταγμα της εξαίρεσης μπορεί να γίνεται και από κάποια μέθοδο που καλείται μέσα στο try block
- Αν υπάρξει εξαίρεση η ροή του κώδικα μεταφέρεται στο catch-block το οποίο χειρίζεται τις εξαιρέσεις

```
try
{
   <Κώδικας πριν>

   <Κώδικας ο οποίος μπορεί να κάνει throw exception>

   <Κώδικας μετά>
}

catch (Exception e)
{
   <Κώδικας που χειρίζεται την εξαίρεση>
   <Χρησιμοποιεί το αντικείμενο e>
}
```

Μέθοδοι που πετάνε εξαιρέσεις

- Το πιο σύνηθες είναι ότι την εξαίρεση την πετάμε σε μια μέθοδο και την πιάνουμε σε μία άλλη.

Μέθοδος που πετάει εξαίρεση

- Σύνταξη

```
ReturnType methodName(argument list) throws Exception
{
        <Κώδικας πριν>
        <Κώδικας ο οποίος κάνει throw Exception>
        <Κώδικας μετά>
}
```

- Αν η μέθοδος πετάξει μια εξαίρεση τότε σταματάει η εκτέλεση του κώδικα στο σημείο που πετάει την εκτέλεση.

```
import java.util.Scanner;

public class DivisionDemoSecondVersion
{
   public static void main(String[] args)
   {
        Scanner keyboard = new Scanner(System.in);

        try
        {
           System.out.println("Enter numerator and denominator :");
           int numerator = keyboard.nextInt(), int denominator = keyboard.nextInt();

           double quotient = safeDivide(numerator, denominator);
           System.out.println(numerator + "/" + denominator + " = " + quotient);
        }
        catch(DivisionByZeroException e)
        {
            System.out.println(e.getMessage( ));
            System.exit(0);
        }

        System.out.println("End of program.");
   }


   public static double safeDivide(int top, int bottom) throws DivisionByZeroException
   {
       if (bottom == 0)
           throw new DivisionByZeroException( );

       return top/(double)bottom;
   }
}
```

Εφόσον έχουμε μία μέθοδο που πετάει εξαίρεση, πρέπει να τη βάλουμε μέσα σε try-catch block

Η εξαίρεση δημιουργείται στην safeDivide αλλά την πιάνουμε και την χειριζόμαστε στην main

Catch or Declare

- Μια μέθοδος η οποία καλεί μια άλλη μέθοδο που πετάει εξαίρεση έχει δύο επιλογές
  - Catch: Να πιάσει και να χειριστεί την εξαίρεση.
  - Declare: Να κάνει κι αυτή throw την εξαίρεση.
    - Αυτό είναι μια μορφή μετάθεσης ευθυνών, αφήνουμε την παραπάνω μέθοδο να χειριστεί την εξαίρεση.
- Αν δεν κάνουμε ένα από τα δύο, ο compiler θα παραπονεθεί.

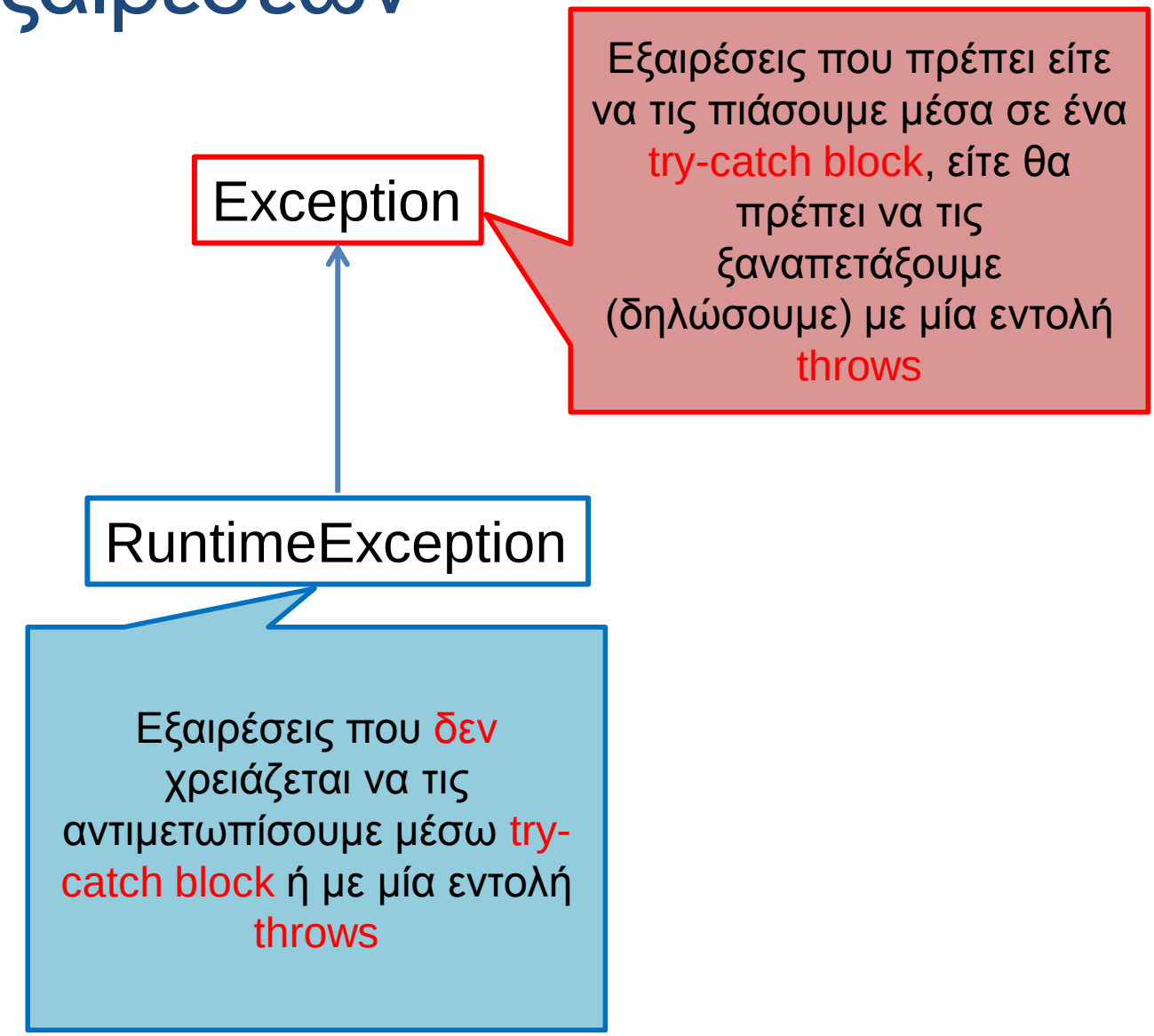
- Εξαίρεση: Runtime exceptions
  - Κάποιες εξαιρέσεις μπορούμε απλά να τις αφήσουμε. Αν συμβούν το πρόγραμμα μας θα τερματίσει με λάθος
  - Π.χ., NullPointerException

```
import java.util.Scanner;

public class DivisionDemoSecondVersion
{
   public static void main(String[] args)
   {
        Scanner keyboard = new Scanner(System.in);

        try
        {
           System.out.println("Enter numerator, den      ator :");
           int numerator = keyboard.nextInt(); i      denominator = keyboard.nextInt();

           int percentage = safePercentage(numerator, denominator);
           System.out.println("percentage  = " + percentage +"%");
        }
        catch(DivisionByZeroException e)
        {
            System.out.println(e.getMessage( ));
            System.exit(0);
        }
   }


   public static int safePercentage(int top, int bottom) throws DivisionByZeroException
   {
       double ratio = safeDivide(top,bottom);
       return (int)(ratio*100);
   }

   public static double safeDivide(int top, int bottom) throws DivisionByZeroException
   {
       if (bottom == 0)
           throw new DivisionByZeroException( );
       return top/(double)bottom;
   }
}
```

Εφόσον έχουμε δεν πετάει εξαίρεση, θα πρέπει να τη βάλουμε μέσα σε try-catch block

Η safePercentage δεν χρειάζεται try-catch block γιατί πετάει κι αυτή την εξαίρεση της safeDivide (declare). Αλλιώς θα ειχαμε compile error.

Τύποι Εξαιρέσεων



```
import java.util.Scanner;
import java.util.InputMismatchException;

public class InputMismatchExceptionDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        int number = 0; //to keep compiler happy
        boolean done = false;

        while (! done)
        {
            try
            {
                System.out.println("Enter a whole number:");
                number = keyboard.nextInt();
                done = true;
             }
             catch(InputMismatchException e)
             {
                 keyboard.nextLine();
                 System.out.println("Not a correctly written whole number.");
                 System.out.println("Try again.");
             }
        }

        System.out.println("You entered " + number);
    }
}
```

Αν και δεν είναι απαραίτητο μπορούμε να πιάσουμε ένα RuntimeException.

Στο παράδειγμα αυτό χρησιμοποιούμε το InputMismatchException για να δημιουργήσουμε ένα βρόχο μέχρι να δωθεί το σωστό input

Το InputMismatchException είναι υπάρχουσα RuntimeException της Java

Χρήση Εξαιρέσεων

- Τις εξαιρέσεις θα τις δείτε περισσότερο όταν θα πρέπει να χρησιμοποιήσετε κάποια βιβλιοθήκη που έχει μεθόδους που πετάνε εξαιρέσεις.
- Στον δικό σας κώδικα έχει νόημα να πετάξετε μια εξαίρεση όταν έχετε μία μέθοδο που δεν ξέρει πώς να χειριστεί ένα λάθος και η απόφαση θα πρέπει να παρθεί σε κάποιο υψηλότερο σημείο του κώδικα που έχουμε περισσότερες πληροφορίες

Προσοχή

- Η εύκολη και τεμπέλικη λύση για μια εξαίρεση είναι να την πιάσουμε και απλά να μην κάνουμε τίποτα, αλλά αυτό είναι κακή προγραμματιστική τακτική.

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 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.

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

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:
  - hasNextLine(): επιστρέφει true αν υπάρχει κι άλλη γραμμή να διαβάσει
  - hasNextInt(): επιστρέφει true αν υπάρχει κι άλλος ακέραιος

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

           <υπόλοιπος κώδικας…>

    }
}
```

STRING PROCESSING

Strings

- Μερικές χρήσιμες εντολές για επεξεργασία Strings
  - lowerCase(): μετατρέπει όλους τους χαρακτήρες ενός String σε μικρά γράμματα.
  - trim(): αφαιρεί λευκούς χαρακτήρες από την αρχή και το τέλος
  - split(RegularExpression): Σπάει το String με βάση το Regular expression, και επιστρέφει ένα πίνακα από Strings με τα κομμάτια.
  - replaceAll(RegularExpression, Character): αντικαθιστά τα κομμάτια που ταιριάζουν σε μια λογική συνθήκη

StringTokenizer

- Σπάει ένα String σε tokens (λέξεις) και μας επιτρέπει να διατρέχουμε τις λέξεις.
  - nextToken(): επιστρέφει το επόμενο token
  - hasTokens(): μας λέει αν έχουμε άλλα tokens

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

```
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