**Parse text in Java**

There are various ways of parsing text. The usual tools are:

* [String.split](http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#split%28java.lang.String%29) methods
* [StringTokenizer](http://docs.oracle.com/javase/7/docs/api/java/util/StringTokenizer.html) and  [StreamTokenizer](http://docs.oracle.com/javase/7/docs/api/java/io/StreamTokenizer.html) classes
* [Scanner](http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html) class
* [Pattern](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html) and [Matcher](http://docs.oracle.com/javase/7/docs/api/java/util/regex/Matcher.html) classes, which implement regular expressions
* for the most complex parsing tasks, you can use tools such as [JavaCC](https://java.net/projects/javacc)

## [Reading files](http://www.vogella.com/tutorials/JavaIO/article.html#example)

To read a text file you can use the Files.readAllBytes method. The usage of this method is demonstrated in the following listing.

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

// somewhere in your code

String content = new String(Files.readAllBytes(Paths.get(fileName)));

To read a text file line by line into a List of type String structure you can use the Files.readAllLines method.

List<String> lines = Files.readAllLines(Paths.get(fileName));

Files.readAllLines uses UTF-8 character encoding. It also ensures that file is closed after all bytes are read or in case an exception occurred.

**Reading and filtering line by line**

The Files.lines method allows read a file line by line, offering a stream. This stream can be filtered and mapped. Files.lines does not close the file once its content is read, therefore it should be wrapped inside a try-with-resource statement.

In the following example unnecessary whitespace at the end of each line is removed and empty lines are filterer.

//read all lines and remove whitespace (trim)

//filter empty lines

//and print result to System.out

Files.lines(new File("input.txt").toPath())

.map(s -> s.trim())

.filter(s -> !s.isEmpty())

.forEach(System.out::println);

The next example demonstrates how to filter out lines based on a certain regular expression.

Files.lines(new File("input.txt").toPath())

.map(s -> s.trim())

.filter(s -> !s.matches("yourregularexpression"))

.forEach(System.out::println);

The next example extracts a line starting with "Bundle-Version:" from a file called "MANIFEST.MF" located in the "META-INF" folder. It removes the prefix and removes all leading and trailing whitespace.

package com.vogella.eclipse.ide.first;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.util.Optional;

import java.util.stream.Stream;

public class ReadMANIFESTFile {

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

String versionString = readStreamOfLinesUsingFiles();

System.out.println(versionString);

}

private static String readStreamOfLinesUsingFiles() throws IOException {

Stream<String> lines = Files.lines(Paths.get("META-INF", "MANIFEST.MF"));

Optional<String> versionString = lines.filter(s -> s.contains("Bundle-Version:")).map(e-> e.substring(15).trim()).findFirst();

lines.close();

if (versionString.isPresent())

{

return versionString.get();

}

return "";

}

}

Create a new Java project called com.vogella.java.files. Create the following FilesUtil.java class.

package com.vogella.java.files;

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.nio.file.StandardOpenOption;

import java.util.List;

public class FilesUtil {

public static String readTextFile(String fileName) throws IOException {

String content = new String(Files.readAllBytes(Paths.get(fileName)));

return content;

}

public static List<String> readTextFileByLines(String fileName) throws IOException {

List<String> lines = Files.readAllLines(Paths.get(fileName));

return lines;

}

**Find Special Characters In A String**

**Using regular expressions to specify tokens**

keyword = begin | end | if | then | else

identifier = letter (letter | digit | underscore)\*

integer = digit+

relop = < | <= | = | <> | > | >=

letter = a | b | ... | z | A | B | ... | Z

digit = 0 | 1 | 2 | ... | 9

#### Write a regular expression to represent all valid identifiers in java language.

**Rules:**  
The allowed characters are:

1. a to z, A to Z, 0 to 9, \_,#
2. The 1st character should be alphabet symbol only.

Program:

import java.util.regex.\*;

class RegularExpressionDemo

{

public static void main(String[] args)

{

Pattern p=Pattern.compile("[a-zA-Z][a-zA-Z0-9\_#]**+**");

Matcher m=p.matcher(args[0]);

if(m.find()&&m.group().equals(args[0]))

{

System.out.println("valid identifier");

}

else

{

System.out.println("invalid identifier");

}

}

}

Output:

E:\scjp>javac RegularExpressionDemo.java

E:\scjp>java RegularExpressionDemo ashok

Valid identifier

E:\scjp>java RegularExpressionDemo ?ashok

Invalid identifier

An easy way is to check if a string has any non-alphanumeric characters.

**Source: (SpecialChars.java)**

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**public** **class** SpecialChars {

**public** **static** **void** main(String[] args) {

String str = "abc$def^ghi#jkl";

Pattern p = Pattern.compile("[^a-z0-9 ]", Pattern.CASE\_INSENSITIVE);

Matcher m = p.matcher(str);

System.out.println(str);

**int** count = 0;

**while** (m.find()) {

count = count+1;

System.out.println("position " + m.start() + ": " + str.charAt(m.start()));

}

System.out.println("There are " + count + " special characters");

}

}

**Output:**

$ java SpecialChars

abc$def^ghi#jkl

position 3: $

position 7: ^

position 11: #

There are 3 special characters

Remarks:

1. "[^a-z0-9]" means anything except (a-z followed by 0-9)
2. [] means a range of characters

#### Extracting/Capturing

Here's a one loop approach using regex. You construct a pattern using your keywords, and then iterate through your dataList and see if you can find a match.

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

List<String> keywords = new ArrayList(Arrays.asList("FIRSTNAME", "LASTNAME", "CURRENCY", "FUND"));

List<String> dataList = new ArrayList(Arrays.asList("HUSBANDFIRSTNAME", "HUSBANDLASTNAME", "WIFEFIRSTNAME", "SOURCECURRENCY", "CURRENCYRATE"));

Set<String> targetSet = new HashSet();

String pattern = String.join("|", keywords);

for (String data : dataList) {

Matcher matcher = Pattern.compile(pattern).matcher(data);

if (matcher.find()) {

targetSet.add(matcher.group());

}

}

System.out.println(targetSet);

}

Results:

[CURRENCY, LASTNAME, FIRSTNAME]

Specific values can be selected out of a large complex body of text. These values can be used in the application.

Sample code

import java.util.ArrayList;

import java.util.List;

import java.util.regex.\*;

public class ExtractDemo {

public static void main(String[] args) {

String input = "I have a cat, but I like my dog better.";

Pattern p = Pattern.compile("(mouse|cat|dog|wolf|bear|human)");

Matcher m = p.matcher(input);

List<String> animals = new ArrayList<String>();

while (m.find()) {

System.out.println("Found a " + m.group() + ".");

animals.add(m.group());

}

}

}

**This produces the following output:**

|  |
| --- |
| Found a cat.  Found a dog. |