

PROGRAMMING ASSIGNMENT 6:

Read Savitch: Chapter 2, 9, String/StringBuffer/StringTokenizer handout

Programming: Name your file for PA6: **PA6.java** in the dir. named **PA6**

DUE: June 5, 2002 at 11pm

Turnin: **turninPA6**

REQUIRED:

This programming assignment will focus on elementary parsing of lines of simple integer arithmetic expressions (**Strings**) with **StringTokenizer** read from a file with a **BufferedReader**, building a resulting output String with a **StringBuffer** and writing the output to a file with a **PrintWriter**, along with basic Exception Handling within an application similar to PA5. The resulting output String will be the expression spelled out in English. A recursive method will be used to convert integers into their equivalent English expression.

Program overview: Report a Usage message if the user has not supplied the correct number of command line arguments. Try to open the input and output files as specified on the command line. Read each line from the input file. Parse each line into a left operand (integer), an operator (one of '+', '-', '*', '/', '%'), and a right operand (integer) using a StringTokenizer object handling the case that the first operand may start with a '-' since that character needs to be used as a possible token delimiter. Tokens may or may not have spaces around them. If the line can be parsed correctly, convert the parsed expression into an English phrase including the result of evaluating the arithmetic expression.

public class PA6

```
// ... Possibly other private instance vars and public static final constants
```

```
private int[] operands = new int[2];
```

```
private char operator;
```

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

```
public void openFiles( String infile, String outfile )
```

```
public boolean parseLine( String line )
```

```
public String convertExpressionToEnglish( int[] operands, char operator )
```

```
public String numToEnglish( long num ) - MUST BE RECURSIVE
```

You will want to look at the on-line Java documentation for the various classes you will use in this assignment:

<http://java.sun.com/j2se/1.3/docs/api/index.html>

main()

Check that there are the correct number of arguments - output a Usage message to the terminal if needed. Try opening the supplied input and output files (**openFiles()**). While there are lines in the input file (**readLine()**), print the line to the output file, parse the line (**parseLine()**) setting the two operands and operator, and if there were no errors as indicated by **parseLine()**'s return value then convert the expression (operands and operator) into English along with the evaluated result of the arithmetic expression (**convertExpressionToEnglish()**) and print to the output file. Also be sure to close() each file stream.

You can handle the various exceptions that may be thrown in these methods here or in the methods where they occur.

openFiles()

Try to open the input and output files. Examples of opening and using file I/O:

```
in = new BufferedReader( new FileReader( infile ) );
```

```
out = new PrintWriter( new BufferedWriter( new FileWriter(outfile) ), true );
```

Check the on-line documentation for the types of exceptions that are thrown if the files cannot be opened and be sure to catch them displaying the exception message (**getMessage()**) and exit.

parseLine()

Create a **StringTokenizer** object with the line. The delimiter characters will need to be the whitespace characters and the valid operators " \t\n+-*/%". We will want to return the delimiter character as a token with each call to **nextToken()** since there may be no spaces to separate the tokens. Since '-' is a token delimiter and also part of a negative integer, we will need to handle this case for at least the first operand. We will also want to eat any extra whitespace characters that may be between operand/operator and any leading/trailing whitespace. Try to convert the first token to an int (**Integer.parseInt()**), the next non-whitespace token whose length is 1 as an operator (char) based on what are valid operators, and the next non-whitespace token as an int.

Assign operands[0] as the left operand (int), operator as the operator (char), and operands[1] as the right operand (int).

If there are more tokens on the line or you encountered any error while parsing the line for the two operands and operator, return false to indicate the line did not parse correctly. Otherwise, return true.

convertExpressionToEnglish()

Use the operands and operator arguments to build the English expression and result of evaluating the expression. You must use a **StringBuffer** object to build (**append()**) the English phrase. Use **numToEnglish()** to convert each integer operand and the result to its equivalent English phrase. See the sample program for examples.

Return the StringBuffer converted to a String (**toString()**).

numToEnglish()

This method must be recursive. Check if the number is negative. If it is, then set the result string to "negative " and change the sign of the negative number to positive for subsequent recursions. Grab the right-most digit of the parameter (%10) and save it in a local variable. Get rid of the right-most digit of the parameter (/10). If there are still more digits in the parameter, recursively call numToEnglish() passing in the modified parameter value appending the return value of the recursive call to the result string. As you pop off the recursive calls, append the English word equivalent of the saved right-most digit to the result string and return this result string. You may use either String concatenation or StringBuffer append()/toString() in this method.

There are many ways one can parse expressions like this. Compilers parse in a much more elaborate fashion than described here. Your error states/messages may differ from the sample program depending on how you parse the expression.

Examples:

% java PA6

Usage: java PA6 input_file output_file

% java PA6 foo

Usage: java PA6 input_file output_file

% java PA6 foo bar

foo (No such file or directory)

% java PA6 infile /dev/kmem

/dev/kmem (Permission denied)

```
% cat infile
```

```
-35 + 903
```

```
55-590
```

```
-323--3
```

```
33 - -33
```

```
% java PA6 infile outfile
```

```
% cat outfile
```

```
-35 + 903
```

```
negative three five plus nine zero three equals eight six eight
```

```
55-590
```

```
five five minus five nine zero equals negative five three five
```

```
-323--3
```

```
negative three two three minus negative three equals negative three two zero
```

```
33 - -33
```

```
three three minus negative three three equals six six
```

```
% java PA6 infile1 outfile1
```

```
% cat outfile1
```

```
55 / 0
```

```
ArithmeticException Error: / by zero
```

```
55 % 0
```

```
ArithmeticException Error: / by zero
```

```
55 55 55
```

```
Illegal operator: 55
```

```
55 + 55 - 55
```

```
Too many tokens on line.
```

```
55 + 5h5
```

```
NumberFormatException Error: 5h5
```

```
55 +
```

```
NoSuchElementException Error: null
```

```
55 +& 55
```

```
NumberFormatException Error: &
```

```
55+&55
```

```
NumberFormatException Error: &55
```

```
55 &+ 55
```

```
Illegal operator: &
```

```
55&+55
```

```
NumberFormatException Error: 55&
```

```
55 // 55
```

```
NumberFormatException Error: /
```

```
55//55
```

```
NumberFormatException Error: /
```

```
55 && 55
```

```
Illegal operator: &&
```