# Quellcode für Beispiel 02

Head:

//takes the first x lines of a file  
public class Head {  
  
 //how many lines should be copied  
 int lineCount = 0;  
  
 public Head(int lineCount) {  
 this.lineCount = lineCount;  
 }  
  
 public Head() {  
 this.lineCount = 9000;  
 }  
  
 //reads from the console  
 public void readFile() throws Exception {  
 try {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
  
 readFile(reader.readLine());  
 } catch (IOException e) {  
  
 }  
 }  
  
 //reads a file  
 public void readFile(String filename) throws Exception {  
 if (this.lineCount > 0) {  
 Scanner scanner = new Scanner(new File(filename));  
 int countedLines = 0;  
  
 //reads line by line  
 while (scanner.hasNext() && countedLines < this.lineCount) {  
 //send one line to next block  
 String line = scanner.nextLine();  
  
 if (!line.equals("") || line.equals("\n")) {  
 System.*out*.println(line);  
 countedLines++;  
 }  
 }  
  
 scanner.close();  
 } else {  
 System.*out*.println("You can't return a negative amount of lines!");  
 }  
 }  
  
}

Tail:

public class Tail {  
  
 //how many lines should be copied  
 int lineCount = 0;  
  
 public Tail(int lineCount) {  
 this.lineCount = lineCount;  
 }  
  
 public Tail() {  
 //it's OVER 9000!!!  
 this.lineCount = 9001;  
 }  
  
 //reads from the console  
 public void readFile() throws Exception {  
 try {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
  
 readFile(reader.readLine());  
 } catch (IOException e) {  
  
 }  
 }  
  
 //reads a file  
 public void readFile(String filename) throws Exception {  
 if (this.lineCount > 0) {  
 Scanner scanner = new Scanner(new File(filename));  
 List<String> strings = new ArrayList<>();  
  
 //reads line by line  
 //has to read all before you can take the last few lines  
 while (scanner.hasNext()) {  
 strings.add(scanner.nextLine());  
 }  
  
 //if the count of the lines the user wants is smaller than the actual size of lines  
 if (this.lineCount < strings.size()) {  
 //takes strings.size - lineCount  
 for (int i = (strings.size() - lineCount); i < strings.size(); i++) {  
 System.*out*.println(strings.get(i));  
 }  
 } else { //user wants more lines than there are  
 for (int i = 0; i < strings.size(); i++) {  
 System.*out*.println(strings.get(i));  
 }  
 }  
  
 scanner.close();  
 } else {  
 System.*out*.println("You can't return a negative amount of lines!");  
 }  
 }  
  
}

LOC:

//returns count of lines, count of words and characters per file  
public class LOC {  
  
 //reads from the console  
 public void readFile() throws Exception {  
 try {  
 BufferedReader reader = new BufferedReader(new InputStreamReader(System.*in*));  
  
 readFile(reader.readLine());  
 } catch (IOException e) {  
  
 }  
 }  
  
 //reads a file  
 public void readFile(String filename) throws Exception {  
 Scanner scanner = new Scanner(new File(filename));  
 int countedLines = 0;  
 int countedWords = 0;  
 int countedCharacters = 0;  
  
 String[] words;  
  
 //reads line by line  
 while (scanner.hasNext()) {  
 //send one line to next block  
 String line = scanner.nextLine();  
 //words get split by space  
 words = line.split(" ");  
  
 for (int i = 0; i < words.length; i++) {  
 //gets the length from one word and adds it  
 countedCharacters += words[i].length();  
 countedWords++;  
 }  
  
 countedLines++;  
 }  
  
 scanner.close();  
  
 printStatistics(filename, countedLines, countedWords, countedCharacters);  
 }  
  
 private void printStatistics(String filename, int lines, int words, int characters) {  
 System.*out*.println("Statistics for " + filename);  
 System.*out*.println("--------------------");  
 System.*out*.println("Lines: " + lines);  
 System.*out*.println("Words: " + words);  
 System.*out*.println("Characters: " + characters);  
 }  
  
}

TreeSize:

public class TreeSize {  
  
 private String format = "|--";  
 private String space = " ";  
 private String pattern = "###,###.###";  
 private DecimalFormat decimalFormat = new DecimalFormat();  
  
 private long tree(String path, int depth) {  
 try {  
 long size = 0;  
  
 //create a new File with the path  
 File folder = new File(path);  
 //gets an array with the files in this path  
 File[] files = folder.listFiles();  
  
 //goes through the files in this directory  
 for (int i = 0; i < files.length; i++) {  
 //the current file is a directory  
 if (files[i].isDirectory()) {  
 String newPath = files[i].toString();  
  
 size += tree(newPath, depth + 1);  
  
 //print the size for the directory  
 System.*out*.println(format + " Current size of "  
 + files[i].toString() + ": "  
 + decimalFormat.format(size)  
 + " Byte");  
 } else { //get the size of one file  
 size += files[i].length();  
  
 for (int j = 0; j < depth; j++) {  
 System.*out*.print(space);  
 }  
  
 if (files[i].isFile()) {  
 //print the size for the current file  
 System.*out*.println(format + " Current size of "  
 + files[i].toString() + ": "  
 + decimalFormat.format(files[i].length())  
 + " Byte");  
 }  
 }  
  
 }  
  
 return size;  
 } catch (NullPointerException n) {  
 System.*out*.println("The path is invalid!");  
 return 0;  
 }  
 }  
  
 public void calculateTree(String path) {  
 System.*out*.println("Directory: " + path);  
  
 long directorySize = tree(path, 0);  
  
 System.*out*.println("Size of directory " + path + ": "  
 + decimalFormat.format(directorySize) + " Byte");  
 }  
  
}

Tests:

HeadTest:

public class HeadTest {  
  
 public static void main(String[] args) {  
  
 }  
  
 //tests for Head  
 public static void test01() throws Exception {  
 Head h = new Head(3);  
 h.readFile("Monika.txt");  
 }  
  
 public static void test02() throws Exception {  
 Head h = new Head(-7);  
 h.readFile("Monika.txt");  
 }  
  
 //test it with a line count that's bigger than the file  
 public static void test03() throws Exception {  
 Head h = new Head(99);  
 h.readFile("Monika.txt");  
 }  
  
 public static void test04() throws Exception {  
 Head h = new Head();  
 h.readFile("Monika.txt");  
 }  
  
}

Tail:

public class TailTest {  
  
 public static void main(String[] args) {  
  
 }  
  
 //tests for Tail  
 public static void test05() throws Exception {  
 Tail t = new Tail(3);  
 t.readFile("Monika.txt");  
 }  
  
 public static void test06() throws Exception {  
 Tail t = new Tail(-77);  
 t.readFile("Monika.txt");  
 }  
  
 //test it with a line count that's bigger than the file  
 public static void test07() throws Exception {  
 Tail t = new Tail(1337);  
 t.readFile("Monika.txt");  
 }  
  
 public static void test08() throws Exception {  
 Tail t = new Tail();  
 t.readFile("Monika.txt");  
 }  
  
}

LOCTest:

public class LOCTest {  
  
 public static void main(String[] args) {  
  
 }  
  
 //tests for LOC  
 public static void test09() throws Exception {  
 LOC l = new LOC();  
  
 l.readFile("Monika.txt");  
 }  
  
 public static void test10() throws Exception {  
 LOC l = new LOC();  
  
 l.readFile("IDoNotExist.txt");  
 }  
  
 public static void test11() throws Exception {  
 LOC l = new LOC();  
  
 l.readFile("IAmEmptyInside.txt");  
 }  
  
}

TreeSizeTest:

public class TreeSizeTest {  
  
 public static void main(String[] args) {  
  
 }  
  
 //tests for TreeSize  
 public static void test12() throws Exception {  
 TreeSize tree = new TreeSize();  
  
 tree.calculateTree("D:\\Studium\\Sommersemester 4\\SWP4VO\\Übungen\\Übung02");  
 }  
  
 public static void test13() throws Exception {  
 TreeSize tree = new TreeSize();  
  
 tree.calculateTree("D:\\IDoNotExist");  
 }  
  
 public static void test14() throws Exception {  
 TreeSize tree = new TreeSize();  
  
 tree.calculateTree("D:\\Studium");  
 }  
  
 public static void test15() throws Exception {  
 TreeSize tree = new TreeSize();  
  
 tree.calculateTree("D:\\Studium\\Sommersemester 4\\SWP4VO\\Übungen\\Übung02\\IAmEmpty");  
 }  
  
}