



## ASSIGNMENT 9

### GENERAL INFORMATION

This assignment is made public on Friday, April 4, 2003. The assignment is due on

Friday, April 11, 1 PM.

Hand in your assignment to the teaching assistant running your lab session.

The first page of your (written) assignment has to contain at least the following information:

- the course name (Introduction to Programming - Concepts and Tools)
- your name and your student number
- name and student number of the fellow student if you submit in pairs
- assignment number

*Please staple your assignment!*

You will get back the graded assignment one week after submission deadline.

### WARM-UP AND SUGGESTIONS FOR FURTHER EXERCISES

As warm-up I suggest that you have a look at the review questions of Chapter 10 of the book (p. 367 of my copy). Before working on question 2 below you should read Section 10.5 of the book.

As to further exercises I suggest the following from Chapter 10 of the textbook (pp. 367–369 of my copy):

- Exercises 1–3 are very simple exercises dealing with writing strings to a file.
- Exercises 4–8 deal with reading and writing (random) integer numbers to files, printing their distribution, etc.
- Exercise 9 can be seen as a preparation to question 2 of this assignment, as it asks you to read data from a text file, modify that input, and write it to another textfile.
- Exercises 14–18 deal with encryption and decryption of text files using different ciphers.

It is up to you whether or not you want to work on these exercises. Those exercises will not be marked.

## QUESTIONS

### 1. (*Directory Browser*)

LEARNING OBJECTIVES: Use of the class `File`, design of a short program.

In this assignment we will write a little program that will allow to browse directories. The program should be run using the line

```
java BrowseDir    or    java BrowseDir pathname
```

In the first case the browser starts in the current directory, in the second the browser starts in a sub-directory of the current one, which is specified by `pathname`. In particular, the first version is equivalent to `java BrowseDir .`, that is, path name is just a dot.

The program lists the directory, and allows to enter a string for further action. The following lists the options and the behaviour of the program:

- `..`: The browser stays in the directory and lists its contents (again).
- `...`: The browser lists the contents of the parent directory (and sets the parent directory as *current directory*).
- `path`: If this path leads to a valid directory or file, then that directory should be listed and that directory becomes current directory. If this path leads to a valid file a message should be printed that a file was selected. If the path does not lead to an existing file or directory then a message like *does not exist* should be printed. In the latter two cases the current directory remains unchanged.
- `*`: If the symbol `*` is entered the program terminates.

A running version of the program can be downloaded from the course homepage as `BrowseDir.class`.

- (a) Download the program and play with it so that you understand its functionality.
- (b) (Compare our class notes.) Write a method with signature

```
static void printDir(File f) throws IOException
```

that takes a file object representing a directory and prints out a message like

```
Current directory now /.automount/robin/root/export/home0/butz/  
teaching/java/assignments/
```

```
.  
..  
week1/  
tio/  
week2/  
TestFile.java~  
supplementary/  
TestFile.java  
TestFile.class  
week3/  
week4/  
week5/  
week6/  
week7/  
week8/  
week9/  
week10/  
week11/  
BrowseDir.java~  
BrowseDir.java  
BrowseDir.class
```

- (c) Write your main program. It should first decide, depending on how the program is started, what the current directory is, either `.`, or the directory given as command line argument to the program. Next a `File` object should be created. Check whether it represents a file (in which case the program should halt) or a directory (in which case the program should print its contents). Wait for new input and handle it accordingly.

WHAT TO SUBMIT: Submit the printout of your code, plus sample printouts of the screen showing usage of the program.

## 2. (*Encryption and Decryption*)

LEARNING OBJECTIVES: Reading from and writing to text files.

(You may want to read section 10.5 of the textbook before working on this question.)

You are supposed to write a program that can encrypt and decrypt text files. For simplicity we will only code files that contain lowercase letters (and punctuation, which we leave unchanged).

In the encryption phase the program will use three files, a source file `source`, a file `pad` containing a randomized pad (code), and a file `target` that will contain the encrypted file. That file can only be decrypted if one has access to the `pad`.

The encryption will roughly work as follows: We will read the source file character by character. If the character is not a lowercase letter we print it unchanged both to `pad` and `target`. If it is a lower case letter, however, we will encrypt it as follows: We will generate a random number (“key”) between 0 and 25 and off-set the read character by the key (with the understanding, that when we off-set the character ‘z’ by 1 that we get ‘a’, ‘z’ by 2 that we get ‘b’, etc.). This is the encoded character which is printed to the `target` file. In addition we have to maintain the pad (i.e., the keys). We could print the key as an integer to the `pad` file, but, to get files of similar type, we will off-set the character ‘a’ by key, and store that information in file code.<sup>1</sup>

Except for the fact that one can still see some structure (like punctuation and spaces) in the encrypted file (`target`) this way of encryption is known to be un-breakable (unless one has the file containing the pad). A pad like the one used here is known as a one-time pad.

Before starting with this program you may want to play around with a solution. Download the file `Crypt.class` from the homepage. This program uses the following syntax.

To code a file you have to give the source file (e.g., `message.txt`, the file where you want to store the code (e.g., `pad.txt`), and a file name for the encrypted text (e.g., `encrypted.txt`). Then you can run the program as

```
java Crypt -e message.txt pad.txt encrypted.txt
```

The program will not allow that you overwrite a file containing a pad. This means if you want to encrypt the message again (generating a different pad in the file `pad.txt`) you first have to save or delete the old version of that file.

To decode a file (like `encrypted.txt`) using a pad (for example `pad.txt`) use the command line

```
java Crypt -d encrypted.txt pad.txt decrypted.txt
```

where now the decoded text will be in file `decrypted.txt`.

In the following we will implement a slightly simplified version of that program.

- (a) Play with the program to get a feel for it. Once you encrypted a file have a look at the pad file and the encoded file using a text editor.
- (b) Generate a short text file `message.txt` that will contain your text to be encrypted. The method should contain only lower case letters and punctuation or white space.
- (c) Write a method with (extended) signature

```
static void encrypt(String source, String pad, String target) throws IOException
```

that will do the encryption. Here `source`, `pad`, and `target` hold the file names for the corresponding files. The method should roughly do the following:

- Declare a `BufferedReader` to read from the source file.
- Declare two `PrintWriters` to write to the pad and target files.

---

<sup>1</sup>For those who know modular arithmetic here comes a different description. Principally we match lower case letters of the alphabet with numbers from 0 to 25. To encrypt a letter `l` we take its numerical representation `l`, generate a (random) key between 0 and 25, and store (the letter representation of) key in the file `pad`, and store (the letter representation of) `(key + l mod 26)` in the file `target`. To decode a letter `c` we take its numerical representation `c`, and add to it `(26 - key)`, which boils down to calculating `(c - key mod 26)`, to retrieve the (numerical representation of the) original letter.

- To handle the encryption we will read the source line by line. If the line is empty we reached the end of the source file and are done (but don't forget to close the files you opened for reading and writing).
- If the read line (i.e., the string) is non-empty we do the following: Determine its length (using for example `line.length`) and read each individual character using the method `charAt(int)`<sup>2</sup>, which can be applied to a string and which is part of the `String` class. To check whether a character is a letter or not use the method `boolean Character.isLetter(char)`. Deal with the character according to the next item.
- If the character is not a letter (i.e., white space or punctuation) print it both to the file containing the pad, and to the target file. If the character is a letter generate a random key between 0 and 25 (including). The character (say `l`) is then coded by

`(char)((l-'a'+key)%26+'a')`

(which has to be printed to the target), whereas the key is printed as

`(char)('a' + key)`

to the file containing the pad.

- Once a line is dealt with we have to print a newline to both the file holding the pad, and to the file holding the output. (Why?)

(d) Write a method

`static void decrypt(String source, String pad, String target) throws IOException`

that will decrypt the file `source` using the pad contained in `pad`, and writes the result to file `target`.

The structure of this method is exactly the same as above. The differences are that we have now two files open for reading (the source and the pad), and only one open for writing. Moreover, instead of encrypting we are decrypting. We read in parallel line by line from both source and pad. If the line we read from the source is empty the decryption is finished. If it is non-empty we decrypt that line character by character using the keys found in the line read from file `pad`. For the decryption note that the key is found as

`key = c-'a'`

where `c` is the character that we read from the pad (giving an integer (!) number between 0 and 25), and the decoding is then obtained as

`(char)((l-'a'+26-key)%26+'a')`

where `l` is the encoded letter.

(e) Write a main program that will first encrypt and then decrypt a text file. In this version of your program you should hard-code the names of the files you are using, i.e., you should not ask for user input to name the actual files used for your message, pad, encoded and decoded text. Those files could be

- `message.txt`
- `pad.txt`
- `encrypted.txt`
- `decrypted.txt`

Concentrate on the two methods above and on handling the reading from and writing to files correctly.

WHAT TO SUBMIT: Submit the printout of your code.

---

<sup>2</sup>To use such and similar methods do not forget to import `java.lang.String`.

3. (*Enhancing the previous program*)

This assignment question is *optional*.

You might want to think about enhancing the usability of your program. The following is a list of suggestions. Think of more!

- (a) Enhance your program so that it can be used with command line input as is the case for the sample solution. (This is, indeed, the way many programs are run in a Linux or Unix environment.)
- (b) Print an message about the correct usage of the program if the arguments do not match (more than 4 arguments, the first not starting with a dash, etc.)
- (c) Handle exceptions. Find out first what kind of exceptions can be thrown and when they occur (for this you have to check the Java documentations and read about the methods/constructors that your are using).
- (d) Check whether the files already exist. You might not want to allow that a file containing a pad is over-riden because then the (one-time) pad is lost and the encoded text cannot be recovered.
- (e) ...