# Week 6 Lab - Encryption

COSC244

## 1 Introduction

In lectures and tutorials, you have seen a number of ways to encrypt and decrypt information. During this lab, you will write programs to implement two of the simplest encryption techniques: Caesar cipher and bit-level cipher.

Both of these programs are short, but need a bit of thought to implement correctly. The Caesar cipher program should take less than 10 lines of code, while the bit-level cipher is closer to 20 lines. However, these numbers are given only as a guide, your solution may be longer or shorter.

There is an extension exercise of implementing either a polyalphabetic or a transposition cipher for those who complete the first two programs and want something more challenging. It is not required to get full marks for this lab.

## 2 Assessment

**This lab is worth 1%.** The marks are awarded for the written answers to the preparation questions (worth 0.5%) and completing two programming exercises (worth 0.5%).

The preparation questions should be answered **before** coming to the lab. They should be stored electronically in a plain text file with a .txt suffix in your ~/244/06 directory. They will be checked by the demonstrators at the start of the lab.

**Make sure you get your work signed off by a demonstrator before leaving the lab.**

## 3 Preparation

### 3.1 Exclusive OR

The bit-level cipher makes use of the exclusive OR operator. This cipher can be used on binary files as well as text files. Let's first determine the symbol for the Java exclusive OR operator.

Open the course web page, click on *Resources*, click on *Learning the Java Language*, click on *Language Basics*, and finally click on *Bitwise and Bit shift Operators*.

**Q1** What is the symbol for the exclusive OR operator in Java?

## 3.2 FileInputStream

Let's look at the input/output classes we will use to read and write data for the second program. Remember that a bit-level cipher is suitable for binary data as well as character data. This means that we cannot easily use the `Scanner` class. We need to use classes that read data byte by byte.

Open the course web page, click on *Resources*, click on *Java API Documentation*, click on *FileInputStream* in the *java.io* package.

**Q2** What is this class meant for?

**Q3** Write a statement that uses a constructor to create a new *FileInputStream* and opens a file for reading.

**Q4** What method do you use to read a single byte? What does it return?

## 3.3 FileOutputStream

Now let's look at *FileOutputStream*. In your web browser, click on *FileOutputStream*.

**Q5** What is this class meant for?

**Q6** Write a statement that uses a constructor to create a new *FileOutputStream* and opens a file for writing.

**Q7** What is the signature of the method which writes single bytes?

**Q8** What method should you call when you are finished writing output?

## 3.4 Command line arguments

You might find the *Command-Line Arguments* section of the Java Tutorial helpful when answering the next question if you haven't used command line arguments before. There is a link to this section under *Resources* on the course web page.

**Q9** Both of the programs you will implement require you to use the command line arguments. To get some practice with this, write a program (you don't have to compile it) which takes two arguments. The first argument should be a word and the second argument should be a number which determines how many times to print out the word.

## 4 Programming Exercises

### 4.1 Caesar Cipher

The first program you write will implement a Caesar cipher. Encrypting data using a Caesar cipher involves replacing each character with a character which is a set distance away (in the character set) from it. For example if you were encrypting the word "and" using a shift of 3, it would become "dqg".

A caveat - Depending on the amount of the shift and the *value* of the character being shifted, it is possible the shifted result would be larger than would fit within a byte. You must correctly deal with this situation. Hint: use modulo arithmetic with your shifting to *wrap around* if the result is larger than an 8-bit number (i.e. $> 255$). Write your solution in a file called `Caesar.java` so that it can be easily tested using the script `caesar-check`.

Since you are reading and writing character data, you can use *System.in* as your *InputStream* and *System.out* as your *OutputStream*. Write a program which takes a command line argument to determine the size of the shift, reads input from *System.in*, and sends the encrypted output to *System.out*. You should be able to encrypt and decrypt data, and check that it worked using the command *diff* like this (or you can run the `caesar-check` script which does basically the same thing):

```
$ java Caesar 3 < input.txt > encrypted.txt

$ java Caesar -3 < encrypted.txt > decrypted.txt

$ diff input.txt decrypted.txt
```

You can put a | character immediately after the > character in the commands above to overwrite an existing file. The *diff* command should not show any differences. If the *diff* program does not find any differences, it does not produce any output and you get the Unix prompt back. If there are differences in the two files, it will list them.

### 4.2 Bit-Level Cipher

Encrypting data using a bit-level cipher involves taking an encryption key, let's say a simple string, and performing an XOR (exclusive OR) operation between each byte in the key and the corresponding byte in the information to be encrypted. If the encryption key is shorter than the data, then just loop back to the start of the encryption key and continue the process until all data has been encrypted. To decrypt the encrypted data, simply repeat the encryption process, since performing two XORs with the same key will return the original value.

A caveat - Think carefully about what the first sentence above means. You are reading data byte by byte and doing an XOR between the input byte and a particular byte of the key. So you need to keep track of where you are in the key.

Another caveat - Think carefully about what the second sentence above means. You will run past the end of the key and need to deal with that situation. We will test with input large enough to go past the end of the key.

Write a program to implement a bit-level cipher. The first argument to the program is the name of a file to use as an input stream. The second argument to the program is the name of a file to use as an output stream. Prompt for the encryption/decryption key and use a *Scanner* class to read the key. You should be able to run your program and check the output like this:

```
java BitLevel input.txt encrypted.txt
Enter key:

java BitLevel encrypted.txt decrypted.txt
Enter key:

diff input.txt decrypted.txt
```

The *diff* command should not show any differences. There is a script called `bitlevel-check` which does basically the same thing. Once you completed both programs ask a demonstrator to check your work.

### 4.3 Extra challenge - Polyalphabetic or Transposition Cipher

This program is not required to get full marks for the lab. It is for those who complete the above tasks and want something more challenging.

Try implementing either a polyalphabetic cipher or a transposition cipher. If you choose the transposition cipher, to keep things simple, do not worry about changing the order of the columns. You probably will not have time to finish this exercise during the lab. If you do get one of these implemented, ask a demonstrator to check that it works correctly.