Text File Input/Output in Java

Text files

- Contain readable characters
- Can be opened with a simple text editor such as Java-RTP or Notepad
- Do not contain any formatting such as fonts, size, colours, or graphics
- Universally use ASCII characters

Text File I/O

- Programs in source code are stored as text files so they can be opened with simple editors
- All our Java source code is stored as text files
- Text files usually (but not always) have the suffix `txt` added at the end of the filename (the extension)

```
students.txt
```

Source code file extensions

Extension	**Language**	**Example**
java	Java	hello.java
c	C/C++/C#	sudoku.c
pas	Pascal	inventory.pas
t	Turing	circle.t

Any extension may be used for our own text files, but `txt` is always a good idea

Steps to create a text file in Java:

- Open file
- Perform read or writes on the file
- Close file

To open the file we need to use `import java.io.*;`

then, inside a method,

```
RandomAccessFile f;
try {
    f = new RandomAccessFile("names.txt", "rw");
}
catch (IOException e) {
    System.out.println(e);
}
```

This code declares `f` as a `RandomAccessFile` which will allow us to access the file from inside the program

The actual opening/creating of the file needs to happen inside the `try-catch` construct

In this program, the file `names.txt` is opened for reading and writing with `"rw"`. If the file does not exist, it is created

To open the file (Cont'd)

Java uses the `try-catch` construct because

- Java is not performing the actual file operations. It is the underlying operating system (Windows, Unix, Linux, OS-X) that does the actual file work

- The Operating System returns error codes to our programs

- We can prevent programs from crashing and give the user meaningful error messages such as `File not found`

To open the file (Cont'd)

Some of the most common file operation errors happen in our Java programs when

- We try to open a file for reading but the file does not exist (read operation)
- We try to create a file but the disk is full (write operation)
- We try to create a file but we don't have permissions in the folder where we want to create the file (write operation)
- We try to overwrite a file that is read-only (write operation)

To write to the file (must be inside the `try-catch` construct)

```
f.writeBytes("Jonathan\n");
```

To read from the file (must be inside the `try-catch` construct)

```
String message = f.readLine();
```

To close the file (must be inside the `try-catch` construct)

```
f.close();
```

Example - Create a file and write to it

The following code creates a file called `countries.txt,` writes a few names, then closes the file.

```
import java.io.*;
public class Geography {
    public static void main(String[] args) {
        RandomAccessFile countries = null;
        try {
            countries = new RandomAccessFile("countries.txt", "rw");
            countries.writeBytes("Canada\n");
            countries.writeBytes("France\n");
            countries.writeBytes("Mexico\n");
            countries.close();
        }
        catch(IOException e) {
            System.out.println("Error opening file");
        }
    }
}
```

Example - Read from a file

The following code opens the file `countries.txt`, reads one line and prints it to the screen. It continues reading and printing lines of the file until the end of the file is reached

```
import java.io.*;
public class ShowCountries{
    public static void main(String[] args) {
        RandomAccessFile countries = null;
        try {
            countries = new RandomAccessFile("countries.txt", "r");
            String line = countries.readLine();
            while (line != null) {
                System.out.println(line);
                line = countries.readLine();
            }
            countries.close();
        }
        catch(IOException e) {
            System.out.println(e); // use e for error message
        }
    }
}
```

Example - Open an existing file and write to it from the beginning of the file

The following code opens an existing file called `countries.txt`, The bytes are written starting at position 0 on top of previously existing bytes. If original length of file is larger than the number of bytes being written, the remaining bytes are left untouched, i.e., they are <u>not</u> overwritten.

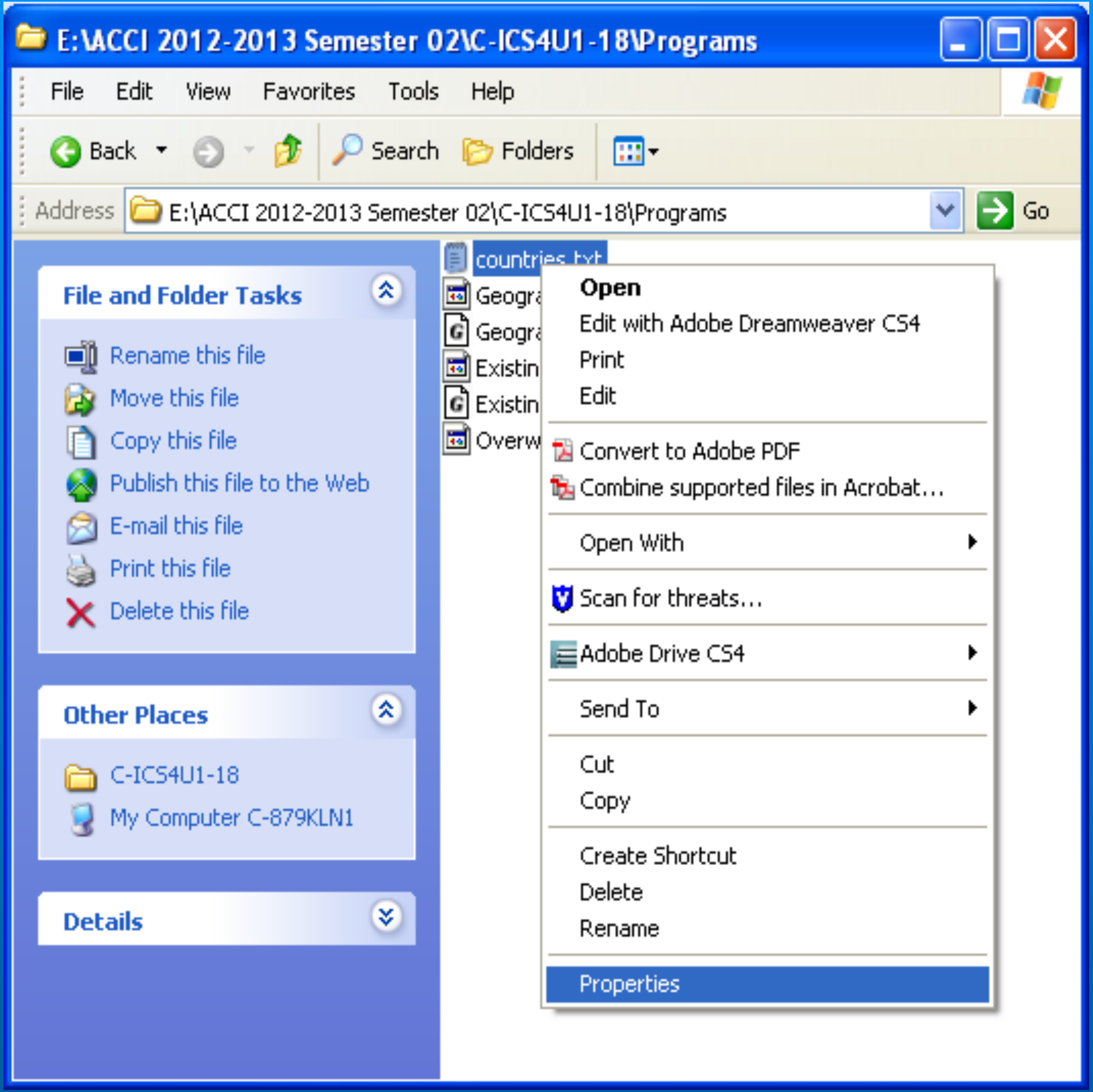
```
import java.io.*;
public class ExistingCountries {
    public static void main(String[] args) {
        RandomAccessFile countries = null;
        try {
            countries = new RandomAccessFile("countries.txt", "rw");
            countries.writeBytes("Australia\n");
            countries.close();
        }
        catch(IOException e) {
            System.out.println("Error opening file");
        }
    }
}
```

Example - Overwrite an existing file

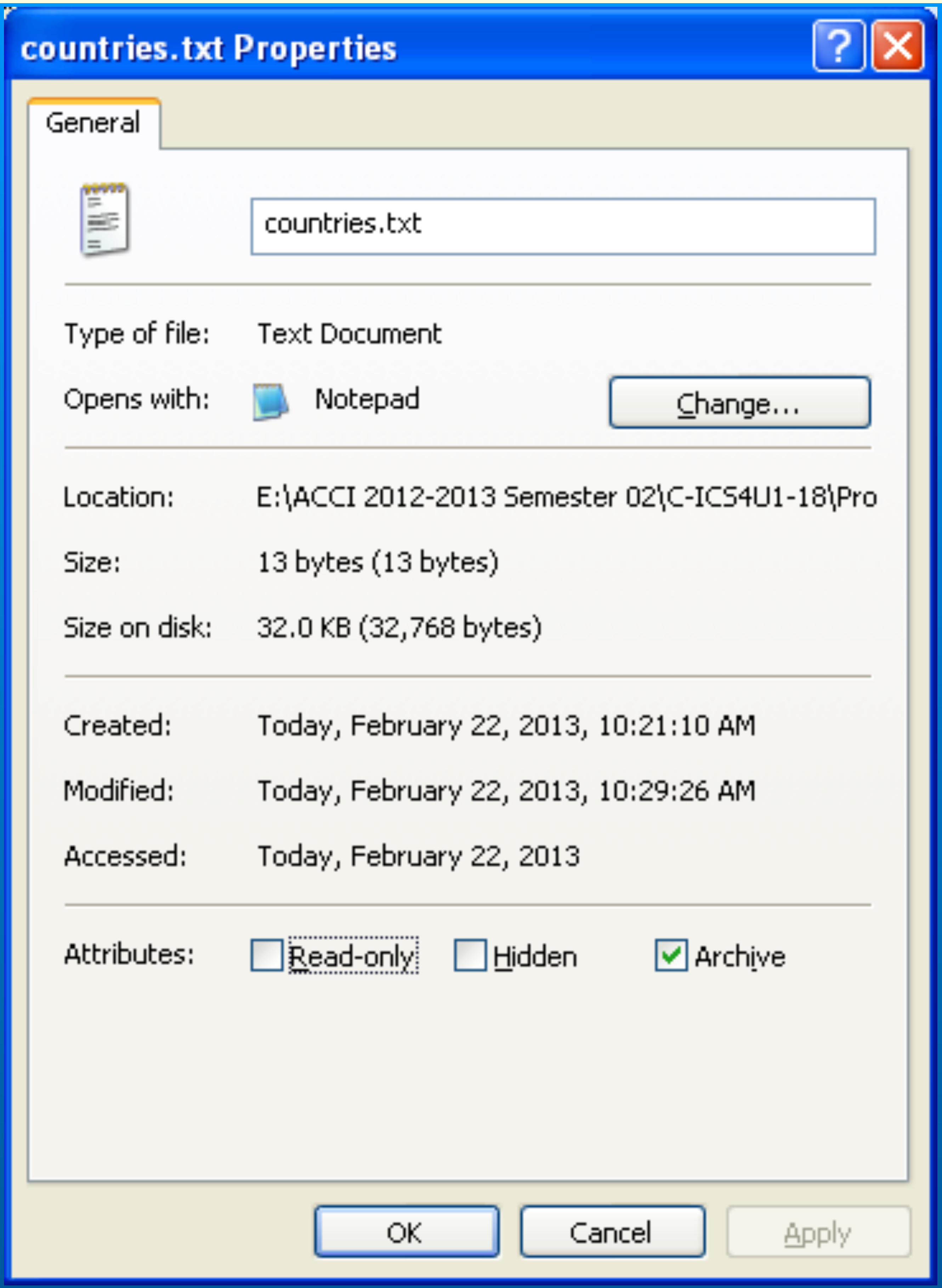
The following code opens an existing file called `countries.txt,` and overwrites the entire file by setting its size to zero.

```
import java.io.*;
public class OverwritingCountries {
    public static void main(String[] args) {
        RandomAccessFile countries;
        try {
            countries = new RandomAccessFile("countries.txt", "rw");
            countries.setLength(0);
            countries.writeBytes("Russia\n");
            countries.writeBytes("Congo\n");
            countries.close();
        }
        catch(IOException e) {
            System.out.println("Error opening file");
        }
    }
}
```

To check the size of the file, right-click on the file and choose properties



The size number of bytes is shown. This number takes into account invisible characters like `'\n'`



countries.txt Properties

General

	countries.txt
Type of file:	Text Document
Opens with:	Notepad
Location:	E:\ACCI 2012-2013 Semester 02\C-ICS4U1-18\Pro
Size:	13 bytes (13 bytes)
Size on disk:	32.0 KB (32,768 bytes)
Created:	Today, February 22, 2013, 10:21:10 AM
Modified:	Today, February 22, 2013, 10:29:26 AM
Accessed:	Today, February 22, 2013

Attributes: ☐ Read-only ☐ Hidden ☑ Archive

Exercise 1

Write a Java program that

Creates a text file called `log1.txt`

Asks the user for a phone number. As soon as the user enters the phone number, the program writes it on the file.

The program closes the file

When you are finished running the program, go to your folder and look for the file `log1.txt` that you just created. Open the file with Java-RTP or another text editor. It should contain the phone number.

Exercise 2

Write a Java program that opens the file `log1.txt` and displays its contents on the system window

Exercise 3

Write a Java program that

Creates a new file called `log2.txt`

Asks the user for a message and writes the message in the file as soon as it is entered

The program keeps asking for a message and keeps on writing each message every time one is entered. Remember to include a `\n` in your `writeBytes()` call

When the word `quit` is entered by the user, the program does not write it. It closes the file and ends

Confirm that the contents are correct by running the program of *Exercise 2* with this new file

Exercise 4

Write a Java method that copies a file to another. The method signature is

```
public static void copyFile(String source, String target)
```

source is the name of the original file and target is the name of the final copied file. You need to supply both of these as parameters. The method `readLine()` returns `null` when the end of file is reached. the sizes of original and copy must be identical. The algorithm for the method is

```
open source file
if source opened successfully {
   open target file

   if target file opened successfully {
      line = read a line from source file
      while end of source file has not been reached {
         write line to target file
         line = read the next line from source file
      }
      close target
      close source
   }
   else {
      close source
      print error message that target file failed to open
   }
}
else {
   print error message that source file failed to open
}
```

Exercise 4 (Cont'd)

Test your method by dowloading the file

```
famous people.txt
```

from Moodle. Run your program and have it create a copy with the name

```
famous people copy.txt
```

The original file is 686 bytes. The copy must be the same size.

Exercise 5

In this exercise we read information from a random position in the file.

Using the Java RTP editor, create a file of 6 records where each one is the exact same size. For example:

```
416-992-2811,ON
905-584-2282,ON
509-282-1181,ON
202-272-4833,DC
315-744-2722,NY
519-483-8293,QC
```

Check the size of the file. In this case it should be 6 records x 16 bytes per record = 96 bytes (remember that each record has an invisible `\n` at the end)

A more realistic view of the file as it is actually stored is:

```
416-992-2811,ON\n 905-584-2282,ON\n 509-282-1181,ON\n202-272-4833,DC\n 315-744-2722,NY\n 519-483-8293,QC\n^Z
```

The very last character is `^Z` which is the end of file character ASCII 26. It is not included in the size of the file.

Exercise 5 (con't)

Here is a more realistic view of the file as it is actually stored,

```
416-992-2811,ON\n905-584-2282,ON\n509-282-1181,ON\n202-272-4833,DC\n315-744-2722,NY\n 519-483-8293,QC\n^Z
```

- File is store sequentially as a *file of bytes*
- All the characters need to be stored, including the new line characters
- The new line character `\n` takes 1 byte of space even though we use two to represent it (the backslash and the letter `n`)
- `\n` is known as a *escape sequence*; it has ASCII code 10
- The last character of the file is control character `^Z` with ASCII code 26, the most commonly accepted end of file marker, but some other operating systems may use `^D`
- There are methods/funcitons in most languages used to check for the end of a file, usually under the acronym EOF. If EOF is used then we do not need to worry about the actual end of file character that a particular operating system uses
- The end of file character is not counted in the size of the file

Exercise 5 (cont'd)

Write a Java method with the signature

```
public static String readRecord(int recordNumber)
```

that uses the method `f.seek()` to move the internal file pointer to the starting position of the record given by `recordNumber`, reads the record into a String and returns it to the calling method. The very first byte is in position zero.

For example,

`readRecord(0)` returns `416-992-2811,ON`
`readRecord(2)` returns `509-282-1181,ON`
`readRecord(4)` returns `315-744-2722,NY`

Summary of methods for Text Files Input/Output

Open/Create

```
RandomAccessFile f = new RandomAccessFile(<Str>, <Str>)
```

Read

```
f.read()
f.readLine()
```
(returns `null` if end of file has been reached)

```
f.getFilePointer()
f.length()
```

Write

```
f.writeBytes(<Str>)
f.write(<byte>)
f.write(<byte[]>)
f.writeInt(<int>)
f.seek(<long>)
f.setLength(<long>)
```

Close

```
f.close()
```

Exercise 5

Write a Java method that compares two files. The method signature is

```
public static void compareFiles(String f1, String f2)
```

The method reads one line at a time from each of the files. As soon as a difference is found the method stops and prints in which line the difference has been found. Let the first line be zero.
The algorithm for the method is

```
open f1
if f1 opened successfully {
   open f2 file

   if f2 opened successfully {
      line1 = read a line from f1
      line2 = read a line from f2
      set line number to 0
      while no differences found and neither end of f1 nor f2 has been reached {
         line1 = read a line from f1
         line2 = read a line from f2
         increment line number
      }
      close f2
      close f1
   }
   else {
      close f1
      print error message that f2 failed to open
   }
}
else {
   print error message that f1 failed to open
}
```

Exercise 6

Write a Java method that sorts a file. The method signature is

```
        public static void sortFile(String info)
```

The method opens the file with the name passed in the parameter `info` and reads every line into an array. The array is then sorted using one of the methods we have learned. The sorted array is then written back to file. The result is a file with the very same name as the original but it is now sorted.

Use the file `famous people.txt` from the Moodle web site as an example. The resulting file will contain the entries sorted by student number.

Exercise 7

Modify exercise 6 so that the resulting file is sorted by the person's name.