# Object-Oriented Development (CIS1056-N) Worksheet 07(A): Files and Exception Handling

## Before You Start

Remember: You are not expected to complete the entire brief within the allotted two hours, but to make a start and continue outside of the class.

Before proceeding with methods, you must be comfortable with program flow control. You should be able to read, understand, and write conditional (if) and iterative statements (while and for). You should also be able to read and write simple Java methods (see last week’s worksheet).

## Introduction

So far, most of our program input has come from the keyboard, with program output being printed to screen. Whilst useful to drive user choice and simple conversion programs, it is time to add files to our repertoire to greatly increase the amount of processing we can do.

As our programs grow in complexity, we are at ever increasing risk of errors and exceptions prematurely halting our program execution. Java’s exception handling, in combination with good programming practices can help mitigate this risk.

**Hint:** In all future exercises you we expect you to make good use of error prevention and exception handling.

## A screenshot of a computer Description automatically generated with medium confidence1. Download the starter solution

FilesAndExceptions.zip is available on Blackboard. Download the file, extract the zip, and open in NetBeans. Confirm the following text files are in the root directory of the project folder:

* mixed.txt
* numbers.txt
* zombie.txt

## 2. Reading zombie.txt

Open the Java file called FileReadingExercise.java. In the main method of this class, write code so that:

1. The user is prompted to type the name of a file at the command line.
2. If the file exists in the current directory, the contents are written to console.
3. If the file does not exist an error message is written to console.
4. This process continues until the user types 'QUIT'.

### Typical Input/Output

Please enter the name of a file or type QUIT to finish

Vampire.txt

Error - Vampire.txt - file does not exist.

Please enter the name of a file or type QUIT to finish

Zombie.txt

Error - Zombie.txt - file does not exist.

Please enter the name of a file or type QUIT to finish

zombie.txt

A zombie is a reanimated human corpse. Stories of zombies originated in the Afro-Caribbean spiritual belief system of Vodou, which told of the people being controlled as laborers by a powerful sorcerer....

Your solution must use the File and Scanner classes. Your solution must catch and handle the FileNotFoundException.

## 3. Reading and Totaling numbers.txt

Open the Java file called FileReadingExercise2.java. In the main method of this class, write code so that:

1. The user is prompted to type in the name of a file at the command line.
2. If the file exists in the current directory, the file is opened for reading.
   1. The text file contains several lines of text. Each line of text begins with a number.
   2. You will read all these numbers and output the total to console.
3. If the file does not exist an error message should be written to console.
4. This process continues until the user types 'QUIT'.

### Typical Input/Output

Please enter the name of a file or type QUIT to finish

fumbers.txt

Error - fumbers.txt - file does not exist.

Please enter the name of a file or type QUIT to finish

numbers.txt

Total: 148

Please enter the name of a file or type QUIT to finish

QUIT

Your solution must use the File and Scanner classes. Your solution must catch and handle the FileNotFoundException.

## 4. Writing to Multiple Files

Create a new Java file called FileReadingExercise3.java. In the main method of this class, write code so that:

1. The user is prompted to type in the name of a file at the command line.
2. If the file exists in the current directory, the file is opened for reading.
   1. The text file will contain several lines of text.
   2. Each line should begin with a number, followed by a name. However, some lines may start with a name followed by a number.
   3. You will read all the numbers from the lines starting with a number and write the total to a text file called 'result.txt'.
   4. If you read a line of text that begins with a name, the line (that is the name and number) should be written to a file called 'errorlog.txt'.
3. If the file does not exist an error message should be written to console.
4. This process continues until the user types 'QUIT'.

### Typical Input/Output

Please enter the name of a file or type QUIT to finish

miped.txt

Error - miped.txt - file does not exist.

Please enter the name of a file or type QUIT to finish

mixed.txt

Written total to result.txt

Written errors to errorlog.txt

Please enter the name of a file or type QUIT to finish

QUIT

To see the contents of the files created use a text editor, or on the Linux console use the command type:

> cat result.txt

Total: 65.41

>cat errorlog.txt

Error at line 3 - Sophie 33.33

Error at line 6 - Candice 12.2222

Your solution must use the File and Scanner classes. Your solution must catch and handle the FileNotFoundException.

## 5. Secret Diary Writer

Create a new Java file called MySecretDiary.java. In the main method of this class, write code so that:

1. The user is prompted to type in the name of a file at the command line – this will be the secret diary - the file is opened for writing (appending)
2. The user is then repeatedly asked to input things that happened to them that day, these will be entries into the secret diary file.
3. Instead of writing the text to the file as the user entered it, the program will ‘encrypt’ the text by modifying each character value so that it is 5 places further along, for example ‘a’ becomes ‘f’, ‘M’ becomes ‘R’, ‘Z’ would be ‘\_’ (underscore) and so on. This is based upon the [Unicode character](https://www.rapidtables.com/code/text/unicode-characters.html) set.
   1. Use the String [charAt()](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#charAt-int-) method to read a single character from the text given by the user using a traditional for loop. Alternatively use the String [toCharArray()](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#toCharArray--) method and use the for/in loop.
   2. Add 5 to this each value and write to the file.

**Hint:** You will have to cast from int to char when writing the character to the file.

This process continues until the user types 'QUIT'.

## 6. Secret Diary Reader

Create a new Java file called MySecretDiaryReader.java In the main method of this class, write code so that:

1. The user is prompted to type in the name of the file at the command line – this will be the secret diary that they wish to open and read. The file is opened for reading.
2. The file is then read and ‘decrypted’ (5 is subtracted from each character value read from the file), and then appended to a String object.
3. Once the file has been completely read and decrypted the complete diary entry is displayed to the user to read.

## 7. Secret Diary Encryption Value

Create copies of your solutions to (5) and (6) – modify them so that the user is now required enter the encryption value used to modify the data written to and read from the secret diary file.

* Try with very a range of positive and negative numbers.
* Try with very small and very large values.

## 8. Secret Diary Further Processing

Create a copy of your solution to (7). Modify the file diary writing code so that instead of immediately writing to file, it builds a string that contains the ‘encrypted’ messages. After the user types QUIT the ‘encrypted’ message is further processed so that if applies the following encoding algorithm.

Where there are runs of the same character value you do the following encoding rules:

1. If there is a 1 character, you simply output that character.
2. If there are 2 of the characters, you write out the 2 characters.
3. If there are 3 or more of the same characters in succession, you write out the number of them then the character.

Examples (using plain text for clarity):

* Hello -> Hello
* Helllo -> He3lo
* Hellllooo -> He4l3o

Note: You should apply this encoding rules to the encrypted message, not the plain text message. However, you should test the encoding method on the plain text first to ensure that it works.

When reading the secret messages in this encoded form, you must reassemble the original message correctly before displaying it back to the user, i.e.

He4l3o would be displayed to the user as Hellllooo.

## Document History

Revision 0 (01-Nov-21): This is the initial version of the 2021/22 exercise.