

# CS 511: Homework Assignment 1

## Due: 22 September, 11:59pm

### 1 Assignment Policies

**Collaboration Policy.** This assignment must be done individually. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions.

**Under absolutely no circumstances code can be exchanged between students of different groups.** Excerpts of code presented in class can be used.

**Assignments from previous offerings of the course must not be re-used.** Violations will be penalized appropriately.

**Late Policy.** Late submissions are allowed. The policy is 2 points off for every hour past the deadline.

### 2 Concurrent File Writing

The idea behind this assignment is to get students familiar with the basics of threads in Java. The task itself is simple, but the implementation does involve some consideration about how threads are to be spawned, and also forces the student to think about the role each thread should play in the overall solution.

Students are asked to implement a method that reads a text file into memory, concurrently rearranges this buffer, and then writes the rearranged buffer into a new output file. Figure 1 depicts this architecture in the case that the rearrangement is performed by 5 threads (Note: this is just an example, your solution may require more or less than 5 threads; please read on). The input and output buffer are different buffers.

In this assignment, we will just be dealing with text files. A file will be seen as a sequence of fixed size “chunks”. In particular, the file size must be a multiple of the chunk size. The name of the file and the size of a chunk is specified by the user in stdin as a command line argument. Here is a sample execution:

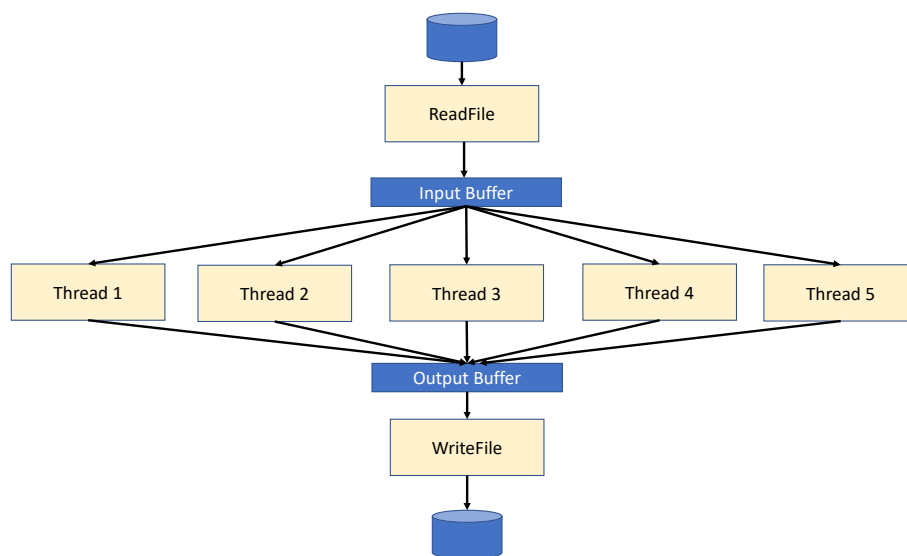


Figure 1: Assignment architecture

```
# java TextSwap 4 letters.txt
```

In the above, **TextSwap** is the main class, 4 is the size of a chunk and **letters.txt** is a text file whose size is a multiple of 4. This file is part of the stub.

The name of the output file is “output.txt”. The number of chunks that make up a file can be calculated by dividing the file size by the specified chunk size.

The order in which a file’s chunks are rearranged is based on a *rearrangement pattern*, which can be seen as a pattern for permuting the chunks. Each chunk is named in order with respect to the alphabet. A file with 5 chunks would have chunks ‘a’ to ‘e’. A possible reordering of this file would be the rearrangement pattern ‘a c b e d’. For example, given the file “AAABBBCCDDDEEE”, a specified chunk size of 3, and the pattern ‘a c b e d’ the output of the program should be “AAACCCBBBEEEDDD”. The rearrangement pattern is input via stdin.

```

-> FileWriteAssignment java TextSwap 4 numbers.txt
Input 5 character(s) ('a' - 'e') for the pattern.
b
a
c
e
d
  
```

Here is another example. Assuming our chunk size is 5 and our input text file contains:

00000111112222233333444445555566666777778888899999

the rearrangement pattern 'j h i e d c b a f g' should produce

99999777778888844444333332222211111000005555566666

## 2.1 File Structure

This assignment consists of the following files:

- `Interval.java`. Declares the `Interval` class.
- `TextSwap.java`. Declares the `TextSwap` class. This class holds the main method from which the assignment is executed.
- `Swapper.java`. Declares the `Swapper` class.
- `letters.txt`. A sample text file.

### 2.1.1 TextSwap

The `TextSwap` class contains much of the logic for the assignment. It has the methods:

- `private static String readFile (String filename) throws Exception`  
This method should read from a specified file by placing it into a `StringBuilder` object and then returning the `toString()` of that object.
- `private static Interval[] getIntervals (int numChunks, int chunkSize)`  
This method returns an array of “Intervals”. An interval is just a pair of integers that specify the start and end index of a chunk. These intervals will be delegated to the appropriate threads to ensure the reordering is proper.
- `private static char[] runSwapper (String content, int chunkSize, int numChunks)`  
This method does much of the actual logic of the class. It creates the intervals, runs the `Swapper` threads, and returns the reordered buffer that will be written to the new file.
- `private static void writeToFile (String contents, int chunkSize, int numChunks) throws Exception`  
This method writes the buffer to a new file.
- `public static void main (String [] args)`  
The main should parse two command line inputs, a chunk size and a filename. The size of the file and the number of chunks should then be calculated, and the new pattern of letters should be read from `stdin`. If the number of chunks is more than 26, then execution should halt with

an error message “Chunk size too small”. If the file size is not a multiple of the chunk size, then execution should halt with an error message “File size must be a multiple of the chunk size”. Note that there may be other methods necessary to complete this class, but they can also be inlined into these methods.

### 2.1.2 Swapper

This class, which should implement the `Runnable` interface, will write to the buffer given its `Interval`. It has the fields `offset`, `interval`, `content`, `buffer`:

```
1 public class Swapper implements Runnable {
2     private int offset;
3     private Interval interval;
4     private String content;
5     private char[] buffer;
6     ...
7 }
```

`Offset`: specifies the starting index in the buffer where the content will be placed. `Interval`: specifies the starting and ending index of the content in the original file that is being swapped. `Content`: the entire original file in a `String`. `Buffer`: The shared `char[]` that the result is being written to.

- `public void run ()`

Write the specified content into the buffer. Helper methods may be used to retrieve the content and insert it into the proper spot in the buffer.

### 2.1.3 Interval

This is exactly what you would expect from any “Pair” class. There is even a `Pair` class in the Java library that can be used instead.

## 3 Your Task

Implement `Swapper.java` and the methods `runSwapper` and `getIntervals` in file `TextSwap.java`.

## 4 Submission Instructions

Submit a zip file named `hw1.zip` through Canvas containing all the files included in the stub but where all required operations have been implemented. Also include your name as a comment in the file `TextSwap.java`.