The University of Birmingham
School of Computer Science


# *Assignment 04*

## 1 Introduction

This assignment will be marked out of 10 and contributes to your module mark (10% of your final module mark).
Deadline for submission is:

**14:00 Friday 13th March**

While the first assignment was about linked lists and their implementation, and the second about using standard Java Collection classes for Lists, Maps, Sets and Arrays, the third about recursion and binary trees, this one is about priority queues, external merge sorting, CSV files and file handling.

CSV files are data files that represent data as rows of "*Comma Separated Values*" (`https://en.wikipedia.org/wiki/Comma-separated_values`). While there are many options and complexities possible in CSV files, we restrict ourselves to a very simple core.

## 2 Marking

- To get full marks, you will have to provide working implementations for all methods where the comment: "`//WRITE YOUR CODE HERE AND REPLACE THE RETURN STATEMENT`" appears so that it matches the specification provided in the corresponding Javadoc comments.
- There will be 1 mark if your submission compiles correctly and can be run and passes the tests provided that already pass in initial version of the assignment files.
- While a set of tests that cover most of the requirements have been provided for you, more tests, and more and alternative sample data files, may be used as part of the marking progress. Therefore you are advised to write your own extra tests.
- If your submission is not structured correctly or does not compile, or crashes or enters an infinite loop before any tests pass, then no marks will be earned.

## 3 Plagiarism

Plagiarism will not be tolerated: it is unfair to the other students and prevents you from learning, and would give you a mark you don't deserve, and qualifications you don't deserve, hence being unfair to Society as a whole too. Please behave well and reward the module lecturer and TA's by submitting your own, hard work.

All submissions will be checked for copying against other submissions and against sources on the web and elsewhere. Any student found to have

When the module lecturer decides that there is evidence of plagiarism, the case will be forwarded to the Senior Tutor, who will look at the evidence and apply penalties and warnings, or, if necessary, forward this to the University.

- University regulations on plagiarism: `https://intranet.birmingham.ac.uk/as/studentservices/conduct/plagiarism/index.aspx`

## 4 Student Welfare

If you miss or are going to miss a deadline, or are getting behind the learning material, for reasons outside of your control, please contact Student Welfare. Occasionally students represent the University in sports, programming competitions, etc., and if this is going to affect your ability to comply with a deadline, Welfare should be informed. It is the Welfare Team, not the teaching team, who can award extensions and cancellations, and devise other measures to cope with adverse situations. It is important to contact welfare as soon as possible when such situations arise.

# 5 Details

There are a number of methods to implement for this assignment, all in `CsvUtils.java`:

- `getStudentID()` and `getStudentName()`: the usual ones to capure your information for marking purposes
- `splitSortCsv(...)`: split a CSV file, which is assumed to be in random order, into a collection of ordered "*runs*": individual CSV files which are ordered and which have, between them, all the rows of the original CSV file.
- `mergePairCsv(...)`: merges two ordered runs into a single (larger) ordered run.
- `mergeListCsv()`: merges a list of ordered runs into a single ordered run by means of repeated calls to `mergePairCsv(...)`

To assist you there is a set of unit tests provided for you in `CvsUtilsTest.java` and a class that provides methods to handle the CSV format required in `CsvFormatter.java`.

A sample method, `copyCsv(...)` has been provided in `CvsUtils.java` to show you an appropriate example of opening files for reading and writing, and reading from and writing to CSV files using the `CsvFormatter` class. Please note: this method is provided **ONLY** as an example of code that correctly reads and writes CSV files. **You should NOT use this method in any of the code that you write.**

You should carefully read through all the code before starting on the assignment.

## 5.1 Paths, Files, Reading and Writing

This assignment requires creating, reading and writing files. Your assignment project directory contains a directory "`data`". This directory contains some CSV files that you should use in development and testing.

**You should only read files from this directory and write files to this directory. The only files you write to should have names that start with "`temp_`"**. The marking software will run your code in a secure sandbox environment and any attempts to read or write outside this directory or to write to a file whose name does not start with "`temp_`" will cause an exception to be thrown and your program to fail.

The `java.nio.file.Path` class represents full and partial file path strings to be handled. By default, when your program starts, it does so with your project directory as its current working directory. Therefore if you give a relative path name for a file you wish to read, e.g. "`data/state-abbrevs.csv`", then it will look for that file path starting at your project directory. You will frequently want to be able to create paths to new files in the `data` folder:

- Given a path variable `dataDir` to a directory, you can construct a path variable to a file given by a `String` filename, `"temp_copy.csv"`, with: `dataDir.resolve("temp_copy.csv")`
- You can also create a `Path` from `String`s using calls like `Paths.get("data")` or `Paths.get("data", "temp_copy.csv")`
- You can get the `String` name of a file (the last component of a path) with `path.getFileName()`. This actually returns a `Path` object with a single component, but the `toString()` method of `Path` will turn it into a string.
- If you have the path of a file, and want to create the path of a new file in the same directory, you can use: `path.resolveSibling("temp_new_file.java")`
- You will need to generate names for your output files that do not clash with your input file names or any other output file names. To do this, make an output file name from the input filename in the following form: if the input filename is `"abc.csv"`, make the output file name be `"temp_00000_abc.csv"`, where the number part can be incremented for each output that you need. If your input file `Path` is in the variable `fromPath`, and your temporary file number counter is in the integer variable `tempNum`, this can be done with:

```
Path tmpPath = fromPath.resolveSibling(
      String.format("temp_%05d_%s", tempNum, fromPath.getFileName())  );
```

### 5.1.1 Scanner and PrintWriter

For this assignment, you should use the `Scanner` class for reading CSV files and the `PrintWriter` class for writing them. This ensures that all unicode characters are handled appropriately.

- You can open a `Scanner` on a file specified by a `Path` object with:
  `Scanner from = new Scanner(fromPath);`

- The `PrintWriter` class does not currently support opening a file from a `Path` object, so you first have to turn the `Path` object into a `File` object:
  `PrintWriter to = new PrintWriter(toPath.toFile()))`

Once you have opened a file, it is important to ensure that you close it:

- Not closing it can mean that your program can crash if you open many files: there is a limit on how many files you can have open at the same time. While this limit is large, it is possible to hit the limit.
- Not closing open files is a memory leak and wastes memory
- Under some circumstances, not closing a file may cause some of the last writes to that file to not be saved to the file on disk

Therefore you should get in the habit of always closing your files. While you can close them directly, you have to ensure that they get closed even if an exception occurs before you make the call to close them. You can handle this with the `try ... catch ... finally` construct, but there is a much easier and more convenient way: the try-with-resources construct:

```
try (  Scanner from = new Scanner(fromPath);
       PrintWriter to = new PrintWriter(toPath.toFile()))
{
  // do your reading and writing here - do not close
}
```

This form of `try` ensures that every resource that is constructed in the "try" part is open within the following curly braces and is automatically closed when the program leaves the braces, whether that leave is caused by an exception, a return, or simply completing the code in the braces. Note that the code in the try part is a series of assignments separated by semi-colons. You can have as many as you like. You can also nest them, if you wish:

```
try (  Scanner from = new Scanner(fromPath)  )
{
    // read from "from" here

    try (  PrintWriter to = new PrintWriter(toPath.toFile())  )
    {
        // Read from "from" and write to "to" here
    }

    // read from "from" here
}
```

## 5.2 Priority Queues and Comparators

A `PriorityQueue` is Java's implementation of the Priority Queue data structure. To use it you need to use a `Comparator` object which tells the `PriorityQueue` how to compare the objects that you put into it. The `CsvFormatter` class contains a nested `CsvComparator` class that is suitable for this purpose. Since it is nested inside `CsvFormatter`, you can only create a `CsvComparator` object from a `CsvFormatter` object as follows:

```
CsvFormatter formatter = ... // get a formatter object
String columnName = "abbreviation"; // the header of the column used for
    comparing CSV rows
CsvFormatter.RowComparator comparator = formatter.new RowComparator(columnName)
```

For this assignment, you will need to put CSV rows into your priority queues. These CSV rows are implemented as arrays of Strings, so, given the `comparator` created above, you can create a suitable new `PriorityQueue` as follows:

```
PriorityQueue<String[]> pq = new PriorityQueue<>(comparator);
```

## 5.3 Testing Arrays for Equality

Do **NOT** try to use the `.equals` method on an array of strings (i.e. a row) to check if the arrays are the same length and have the same strings: it sounds like it should work but, for good reasons, it is unlikely to do what you expect. The way to do what you want is to use `Arrays.equals()`.

```
String[] row1 = ... // get a CSV row
String[] row2 = ... // get a CSV row
```

```
if (row1.equals(row2))    // DO NOT DO THIS - IT IS VERY UNLIKELY TO DO WHAT YOU
    EXPECT
    ...

if (Arrays.equals(row1, row2)) // This does do what you almost certainly want
    ...
```

### 5.4 Final Instructions

- The precise information about the behaviour required of the missing code is detailed in the Javadoc comments in the files.
- You should modify the methods `CsvUtils.getStudentId()` and `CsvUtils.getStudentName()` in `CsvUtils.java` to return your student id and name.
- You should **NOT** add any further class variables to `CsvUtils.java`, or change the types or visibility of the existing class variables or methods nor modify any of the other files in the `main` part of the src tree.
- You can add extra **private** methods in CsvUtils.java if you like but please note that they are not needed and my solution does not do so.
- You can (and should!) modify the `CsvUtilsTest.java` file and you can add any other test files as you please. Your test files are for your own use and should NOT be submitted.
- You should not modify the package structure or add any new classes that `CsvUtils.java` depends upon to run. Your final submission will be purely the `CsvUtils.java` file, so any modifications outside that file will not be considered and the `CsvUtils.java` file that you submit must work with the other files as they currently stand in the assignment structure.
- You should not use any print statements to standard out or standard error streams: if you want to have some debug output, use the logging calls for Log4j. The logging system will be switched to disable log output when your submission is run for marking purposes.
- For marking purposes, your code will be compiled and executed against a test set in a secure sandbox environment. Any attempts to break the security of the sandbox will cause the execution of your program to fail. This includes infinite loops in your code.
- **Your code may read any file in the "`data`" sub-directory of your project directory, and create and write any file there so long as the file name that you use starts with "`temp_`". The secure sandbox environment that will be used in marking your program will cause your program to fail if it trys to read or write any file that does not match this requirement.**
- When you have completed your code changes to your satisfaction, submit your `CsvUtils.java` file. It does not matter what name this file has in your submission. On marking, whatever filename your submission has, it will be renamed to `CsvUtils.java` before marking.