CS2310 Data Structures and Algorithms with Java
Coursework 2017/8

This coursework is designed to assess your achievement of the following learning outcomes of the module:

- Select and apply suitable Abstract Data Types (ADTs) within a principled software development process and ADT implementations, based on appropriate criteria.
- State Java language support for ADTs and use them for software development.

1 MTR Network Info Centre

> *"Mass Transit Railway (MTR) is a major public transport network serving Hong Kong. . . . it consists of heavy rail, light rail, and feeder bus service centred on an 11-line rapid transit network serving the urbanised areas of Hong Kong Island, Kowloon, and the New Territories. The system currently includes 218.2 km (135.6 miles) of rail with 159 stations, including 91 heavy rail stations and 68 light rail stops."*
>
> (Wikipedia 2017)
>
> Source: `https://en.wikipedia.org/wiki/MTR`

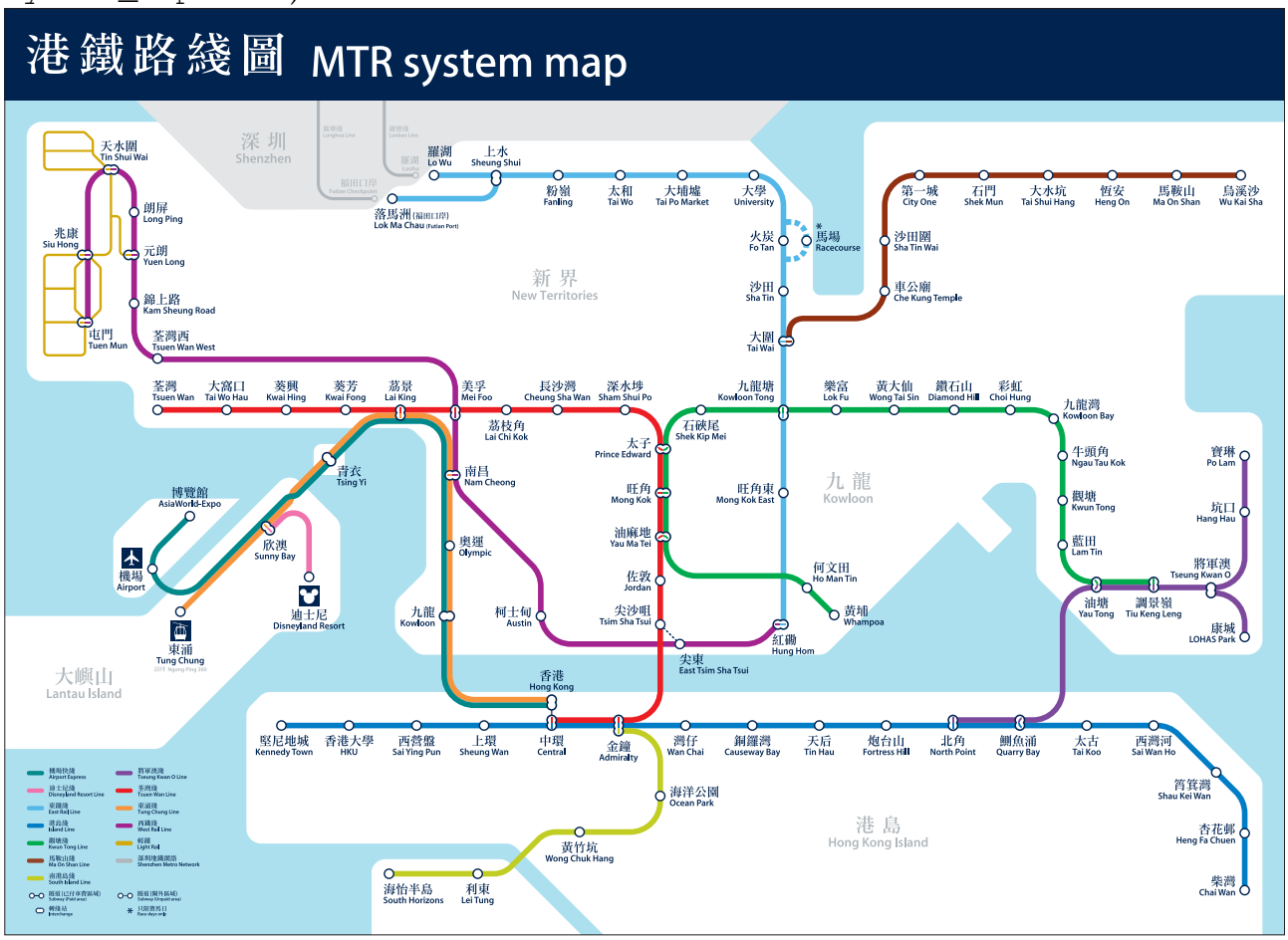
Similar to the London Underground, MTR is made up of a set of interconnected lines. Each MTR line has at least two termini (i.e. the stations at the two ends of a train route). While most lines in the network contains a single sequence of stations, some lines (e.g. `Tseung Kwan O Line` and `East Rail Line`) have branches along the line. Figure 1 shows a map of the MTR system. While theoretically, once entered the MTR network, an MTR passenger can get to anywhere within the network. However, navigating within the network (especially when one is unfamiliar with the network) can be fairly tricky. It is therefore helpful if there would be an Information Centre available to help passengers learn more about the network.

1.1 Your Task

Develop a standalone Java application which works as a single-user prototype application that acts as an MTR Information Centre. This application enables a user to:

- find out which station is a terminus in an MTR line;
- list all known stations in the MTR network;
- find out the directly connected lines for a given MTR line; and
- find a path between two stations.

Figure 1: MTR System Map (`http://www.mtr.com.hk/en/customer/services/system_map.html`).



This prototype application expects to work with a subset of lines with the MTR network and these lines are listed in a text file named `MTRsystem_partial.csv`. Each line in this Comma-Separated-Value (CSV) file contains information about one MTR line, with branching information removed. Each attribute in a data line is separated by comma. The first attribute corresponds to the name of the line, the rest of the attributes correspond to the stations along the line. For example, the information about the `Airport Express` line is shown in the file as:

```
Airport Express,AsiaWorld-Expo,Airport,Tsing Yi,Kowloon,Hong Kong
```

with `AsiaWorld-Expo` and `Hong Kong` being the two termini of the line and there are five stations in this line. The text file `MTRsystem_partial.csv` is available from Blackboard (`http://vle.aston.ac.uk/`).

Your task is to use J2SDK 8 to design and implement the intended MTR Network Information Centre application. This application is expected to work with the subset of MTR lines as shown in `MTRsystem_partial.csv` and there are 11 lines of data in this file.

1.1.1 Functional Requirements

1. A user can *list* all stations that are termini of some MTR lines. For example, `Tuen Mun` and `Hung Hom` are two termini in the `West Rail Line`.

2. A user can *list* all stations in a specified MTR line. The stations in the line are listed in their respective order. For example, the `Disneyland Resort` line may be listed as:

   ```
   Disneyland Resort: Sunny Bay <-> Disneyland Resort
   ```

3. A user can *list* all lines that are directly connected with a specified MTR line. For example, `Island Line` is connected with the following lines directly:

   ```
   Tsuen Wan Line
   South Island Line
   Tseung Kwan O Line
   ```

4. A user can *find* a path between two specified stations. For example, a route between the stations `Airport` and `Disneyland Resort` could be:

   ```
   Airport -> Tsing Yi -> Sunny Bay -> Disneyland Resort
   ```

1.1.2 Non-functional Requirements

1. The processing of **each** query *must* be done in linear time (i.e. $O(n)$) or less. Depending on the query, n may be the number of lines or the total number of stations in the given MTR network.

2. A user will interact with the system through the given Text-based User Interface (TUI), with all output being displayed by the standard output stream.

3. The display of results for each query must be clear and easy to understand.

4. The application *must* be robust and display appropriate messages should any run time errors (e.g. no result found) occur.

2 Further Implementation Hints

1. The given data file uses the standard ASCII character set which is known as `java.nio.charset.StandardCharsets.US_ASCII` in Java.

2. You might like to consider using some of the following facilities provided by J2SDK:

   - IO facilities:
     - `java.util.Scanner`
     - `java.io.BufferedReader`

  - `java.io.File`
  - `java.io.InputStreamReader`
  - `java.io.FileInputStream`
  - `java.nio.charset.StandardCharsets`
  - `java.nio.file.Files`
  - `java.nio.file.Paths`
- facilities from the JCF in `java.util`:
  - Lists: `List`, `ArrayList`, `LinkedList`
  - Maps: `Map`, `Map.Entry`, `HashMap` and `TreeMap`
  - Sets: `Set`, `SortedSet`, `HashSet` and `TreeSet`
  - `Collection`, `Collections`
- facility for representing a sequence of characters which supports efficient operations: `java.lang.StringBuilder`

  String concatenation is the usual way to form a new string by putting several existing strings together. However, when the resulting string becomes very long, this way of concatenating strings will become extremely inefficient.

  When dealing with long strings `java.lang.StringBuilder` would be more efficient. Method `append` in a `StringBuilder` object works well for extending strings in standalone, single-threaded Java applications.

3. To prepare the data for the MTR Network Information Centre application to use, you are likely to need to tokenise each data line after reading it from the data file. You may of course use the old-fashioned `java.util.StringTokenizer` to perform tokenisation. However, as `StringTokenizer` is "*a legacy class and its use is discouraged in new code*" (Oracle 2016), you are encouraged to use the tokenisation facilities in `java.util.String` or the regular expression facilities supported by the `java.util.regex` package. For example, the following Java code:

```
String[] result = "this is a test".split("\\s");
for (int x=0; x<result.length; x++) {
    System.out.println(result[x]);
}
```

prints the following output:

```
this
is
a
test
```

(Source: `http://docs.oracle.com/javase/8/docs/api/java/util/StringTokenizer.html`)

To find out more about how to use regular expression in Java, see:

- `java.util.regex.Pattern`: Define a regular expression (`http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html`)
- `java.util.regex.Matcher`: Perform match operations on a character sequence by interpreting a regular expression (`http://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html`)

4. To evaluate the time efficiency of your algorithm(s), you might like to time some of the tasks performed by your application, e.g.:

   - How long does it take to initialise the application?
   - How long does it take to process a user query?

   You may use the following skeleton Java code for this purpose:

```
long startTime = (new Date()).getTime();
// the process to be timed (eg a method call)
long endTime = (new Date()).getTime();
long elapsedTime = endTime - startTime;
// output the recorded execution time in milliseconds
```

5. To facilitate your system testing, you might like to implement some tester classes or methods, e.g. using `JUnit`. This would enable special tester objects to be created for *simulating the behaviours of various test cases*, hence reducing the time required to repeatedly enter the same set of data into the system after each modification made to the system. However, *the implementation of tester classes/methods is* NOT *a requirement for this coursework*.

3 Mark Distribution

- System design expressed as a detailed UML class diagram(s). (20%)

  Hint: *In your UML class diagram(s), you will need to include detailed information, i.e. fields and methods, about each class in your Java application. Do not simply use* `Eclipse` *to generate a class diagram because some notations used in the* `Eclipse` *plugin do not conform to the UML standard adequately and hence making your design unclear.*

- Your implementation in Java. (70%)

  1. Ability to meet the stated requirements (40%), e.g.:
     - List all termini (6%)
     - List all stations in a line (6%)
     - List all lines that are directly connected with a specified line (8%)
     - Show a path between two stations (12%)

2. Program design and algorithm(s) used (30%)

   Credits will be given to:

   - appropriate use of collection classes in JCF, including setting up each collection object with appropriate initial capacity so as to avoid runtime resizing;
   - appropriate use of other standard Java classes;
   - appropriate use of algorithm(s) for processing and indexing the given data;
   - a robust, generic, efficient and easily-extensible computer system which addresses the requirements of the given scenario.

- Appropriate comments (i.e. documentation, block and line comments) for your implementation and the overall layout and presentation of your programs (eg whether your code has been indented appropriately, whether the naming of variables and classes are meaningful and conform to the standard). (10%)

4 Further particulars

- This coursework is a piece of *group work*, with each group having **no more than 3 members**.
- You should use suitable collection classes in JCF and standard Java packages for your implementation whenever appropriate.
- The API specification for the Java 2 Platform, Standard Edition, version 8 is available at `http://docs.oracle.com/javase/8/docs/api/`
- The use of methods or any programming routines that are not from the J2SDK 8 library, nor are they written by you, in your submission is **not** permitted.

  N.B.: *Submissions which fails to comply with this requirement will be reported to the Associate Dean of Undergraduate Programmes as a suspected case of plagiarism.*

- In your submission, the use of methods or any programming routines that (1) are not given as part of the coursework specification, (2) are not from the J2SDK library, or (3) are not written by you is **not** permitted.

  N.B.: *Submissions which fails to comply with this requirement will be reported to the Associate Dean of Undergraduate Programmes as a suspected case of plagiarism.*

- Your programs must be compilable and executable by J2SDK 8 under command line prompt, without the use of `Eclipse` or any other IDEs.
- Uncommented code will **not** be accepted.

N.B.: *Do not write too much comments neither.*
It is a bad practice to write one line comment for each line of code.

- Your submission must be in `zip` format. Other forms of archives, e.g. `rar`, `tar`, etc, will receive a **penalty** of 5 marks.
- You **may** be required to give a demonstration of your application.

5 References

MTR. (2017), *MTR System Map*. [Online] Available at: `http://www.mtr.com.hk/en/customer/services/system_map.html` [accessed 15 Oct 2017].

Oracle (2016), *Java™ Platform, Standard Edition 8 API Specification*. [Online] Available at: `http://docs.oracle.com/javase/8/docs/api/` [accessed 17 Oct 2016].

Wikipedia. (n.d.), *MTR*. [Online] Available at: `https://en.wikipedia.org/wiki/MTR` [accessed 15 Oct 2017].

This is an assessed exercise which contributes to 20% of the module assessment.

Deadline: 14-12-2017 23:59 (Thursday)

An electronic submission **must** be made to Blackboard **before** 00:00.

The *cut-off date* for receiving coursework submission is *21-12-2017, 23:59*.

N.B.: *Check your submission on Blackboard to ensure that the correct set of files have been submitted. Make sure that you pressed the* SUBMIT *button to complete your submission. Simply uploading your files to Blackboard will* **not** *cause your files to be submitted.*
You are advised to take into account of potential disruption, e.g. network congestion and even network failure. Hence, you are **strongly** *advised to finish all necessary preparation for your submission in good time, rather than leaving it to the last minute.*

Submission details: Submission Procedure:

1. Create a new folder named after the login name of a group member, e.g. if one of your group members has the login name `precious`, your folder should be named as `precious`.
2. Copy, to this new folder, the followings:

(a) your UML class diagram(s) in **PDF** format,

(b) sample output of your application (in plain text format, i.e. files with `.txt` extension), and

(c) your `Eclipse` project folder for this exercise which contains Java programs ONLY (i.e. files with `.java` extension). The project folder structure should be left unchanged.

Make sure that you have REMOVED ALL `.class` *files, Java doc and all hidden files and folders, eg* `.metadata/`, `.project`, *etc.*

3. Create a `zip` archive of this folder (e.g. using `winzip`).

   The archive **must** *be named after the login name used in Step (1).* For example, if the login name used above is `precious`, the name of your zip file will be `precious.zip`.

4. Submit, to Blackboard, the created `zip` archive.
   Other forms of archives, e.g. `rar`, `tar`, *will receive a penalty of* 5 *marks.*

Failure to adhere to the required submission format will be penalised substantially.
Standard lateness penalty[a] *of 10% of the awarded mark for each working day that the piece of work was submitted after the formal deadline will be applied to* every item in each submission.

[a] For more information about lateness penalty, see Chapter 6 of Assessment Regulations and Policies (REG/15/509) which is available at `http://www.aston.ac.uk/EasySiteWeb/GatewayLink.aspx?alId=33431`.