CS3310 – Spring 2014 **Asgn5 (Shortest Path)**
Kaminski **DrivingApp (for Europe)**

******************************** **OVERVIEW** ********************************

DrivingApp finds the shortest path on a road map for designated start and destination city pairs. (A series of pairs are stored in a batch file rather than entering them interactively). Dijkstra's Minimum Cost Path Algorithm is used.

The map is stored as a graph, implemented as an **EXTERNAL ADJACENCY MATRIX** (a binary file). **SetupUtility** builds it from raw, linearly-formatted data. This utility also sets up a CityName file to facilitate user interaction. To aid the developer during testing, this module also "pretty-prints" the binary graph file and the CityName file out (to TheLog file) after they're built.

**************************** **PROJECT STRUCTURE** ****************************

2 options (your choice):

A. DrivingApp and SetupUtility are 2 separate stand-alone programs. You (as human) are the "Main" (controller) who runs these programs manually, specifying interactively (from the console, when prompted) what the fileNamePrefix is.
B. DrivingApp and SetupUtility are 2 separate procedural classes (in separate code files). The Main program is the controller *(as we did for A1-A4)* which runs the "main's" in these 2 classes (drivingAppMain, setupMain), sending in the fileNamePrefix.

Details as to what the controller (you or Main) does will be provided in the DemoSpecs.

************************ **2 main PROGRAMS/MODULES** ***********************

<u>SetupUtility</u> *(OOP optional - good modularization is required)*

- Get fileNamePrefix (via Console for option A, as a parameter for option B) to know which map data to use – options include **Europe** & **Other**.
- Create **?Roads.bin** file and **?CityNames.txt** files from **?MapData.txt** file's data
  (where ? is the fileNamePrefix)
  1. create an INTERNAL adjacency matrix (2-dimensional array) & fill it with data
     You MUST USE MY IN-CLASS APPROACH of 3 possible matrix values:
     - **"infinity" (implemented as short's maxValue) for non-edges**
     - **0's in diagonal**
     - **actual edge values read in from the ?MapData file**
  2. write out this the Roads file and the CityNames file
- Pretty-print ?Roads and ?CityNames files to **TheLog.txt** file

<u>DrivingApp</u> (**MUST use a proper OOP/modular approach**) – the algorithm:

- Declare objects (use these names):
  - **map** – handles accessing of Roads and CityNames files as well as when parts of this data are loaded into memory *[see more below]*
  - **ui** – handles all accessing of `?CityPairs.txt` file and The`Log.txt` file
  - **shortestPath** – handles all path finding (i.e., Dijkstra's Algorithm) including reporting the trace and answers *(by calling ui's writeThis method)*
- Get fileNamePrefix (via Console for option A, as a parameter for option B) to know which map data to use – options include **Europe** & **Other**.
- Loop til ui indicates NO moreTrans

```
{   ask ui for the 2 cities (getStartCityName and getDestinationCityName)
    ask map for the 2 cities' NUMBERS (separately) given the above names
    ask shortestPath to findPath (given the above 2 city numbers)
            [and it'll need to access ui and map objects]
            which will find the answer & the trace, having ui write these to TheLog
}   // may have to change this to a process-read loop structure instead, depending on. . .
```

- Call finishUp methods for the 3 objects

********************** **3 CLASSES (used by DrivingApp)** ***********************

- **Map** – handles access of Roads & CityNames data (some from file, some in memory)
  - N & CityNames are loaded into memory when the file is first opened.
    [NOTE: Roads data is NEVER LOADED INTO MEMORY
    – distance values are always accessed directly from the Roads FILE.]
  - some public service methods needed:
    - constructor – since files are involved and need opening (and certain things need to be loaded into memory)
    - finishUp – since files are involved and need closing
    - **whatsCityName**(short cityNumber)
      Does direct address on the INTERNAL cityNames array
    - **whatsCityNumber**(string cityName)
      For now, just does linear search of INTERNAL cityNames array
    - **getRoadDistance**(short cityNum1, short cityNum2)
      Does **<u>RANDOM ACCESS using the EXTERNAL Roads.bin FILE</u>**
  - some private methods needed:
    - **loadCityNameArray**

[NOTE: the outside world (i.e. DrivingApp, ShortestPath, UI) does NOT know HOW the graph is implemented or accessed inside this class or inside the method bodies - for example, whether things are internal or external, or whether the graph is an adjacency matrix or adjacency lists, or what the structure of cityNames is like an unordered list or a hash table or . . ., etc.]

- **UI** – handles all accessing of `?CityPairs.txt` file and `TheLog.txt` file
  - constructor and finishUp must open/close appropriate files
  - some public service methods needed for transactions
    getStartCityName   getDestinationCityName   moreTrans
  - some public service methods needed for logging
    writeThis(string)

  [NOTE: unlike A1-A4, for this project the input and output batch files are both handled within This SINGLE class rather than separate TransData and TheLog classes]

- **ShortestPath** – handles all path finding (i.e., Dijkstra's Algorithm) and reporting the trace and answers to UI's TheLog file (actual writing done by ui.writeThis)
  - stores: 3 working-storage arrays,
    & trace array/list (STORES TARGETS **IN THE ORDER THAT THEY ARE SELECTED**)
  - public service method needed (others, as needed):
    - findPath (given startNumber and destinationNumber)
  - private service methods needed (others, as needed)
    - initialize the 3 working-storage arrays
    - search
    - reportAnswer which includes both the path distance & the actual path as a list of city NAMES (not numbers)
    - reportTraceOfTargets including both the trace list of cities (as NAMES not numbers) AND the number of Targets

  [NOTE: the outside world (DrivingApp, Map, UI) doesn't care how the shortest path answer is found, what algorithm is used, what working storage is needed in this class, whether there's a search of some database or file to find the answer, a requestToTheCrowd request on the internet or an algorithm for calculating it.

  *[NOTE: INSIDE THIS CLASS, **YOU MUST USE**: **Kaminski's pseudocode version** of Dijkstra's Shortest Path Algorithm - where you ADD THE TRACE part]*

************************* **FILE DESCRIPTIONS - MINE** *************************

`?MapData.txt` *(includes <CR><LF>'s)*

1. Header Line: U or D (for Undirected or Directed Graph), a space, then N
2. CityName Lines (the graph node names):
   N variable-length lines of names for nodes 0 through N-1
3. RoadDistance Lines (the graph edge weights):
   A variable number of lines containing individual edge data in the form:
   cityNumberA space cityNumberB space distance (i.e., edgeWeight)
   [NOTE: For Undirected graphs, ONE line describes both A-to-B and B-to-A
   For Directed graphs, ONE line describes JUST A-to-B]

NOTE: lines starting with a % are comment lines – ignore these
NOTE: Node numbers start at 0 (not 1)
NOTE: cityNames area always single words (e.g., Grand Rapids would be GrandRapids)

`?CityPairs.txt` *(includes <CR><LF>'s) [EuropeTrans and OtherTrans versions]*

- Each line contains a pair of city NAMES: startCityName space destinationCityName
- cityNames area always single words (e.g., Grand Rapids would be GrandRapids)

************************* **FILE DESCRIPTIONS - YOURS** *************************

`?Roads.bin` *A binary, random access file with no <CR><LF>'s and no field-separators.*
Created by SetupUtility.

1. Header Record: N (a short)
2. RoadDistance Matrix: N x N (all short's) (i.e., 0's, "infinity's", actual edge values)

`?CityNames.txt` *A text file (strings or charArrays, ASCII or Unicode) of lines.*
Created by SetupUtility.
Loaded into memory (array) by Map's loadCityNameArray
when DrivingApp declares that object

`TheLog.txt` FROM SetupUtility's prettyPrint method

[NOTE: Data not necessary correct below – I'm just showing the formatting]

```
Map Data:  Europe   Number of cities:  19
Amsterdam
. . .
Warsaw
       0    1    2    3    4    5    6    7    8    9   10  . . .    18
   -----------------------------------------------------------------
  0|    0  512  123 1321  312  432  567 1111 1122  211 1321  . . .  1104
  1| 1133    0  444  449  512 1212  654  102  109 1001 1010  . . .   345
. . .
18|  123  234  345  456  567  678  789  890  111  222  333  . . .     0
```

`TheLog.txt` FROM DrivingApp's ShortestPath class's calling ui.WriteThis method)

[NOTE: Data not necessary correct below – I'm just showing the formatting]

```
#   #   #   #   #   #   #   #   #   #   #   #   #   #   #   #
Amsterdam (0) TO Kalamazoo (-1)
ERROR - one of the cities is not on this map

#   #   #   #   #   #   #   #   #   #   #   #   #   #   #   #
Kalamazoo (-1) TO Amsterdam (0)
ERROR - one of the cities is not on this map

#   #   #   #   #   #   #   #   #   #   #   #   #   #   #   #
Paris (13) TO Copenhagen (6)
DISTANCE:  907
PATH:  Paris > Brussels > Amsterdam > Hamburg > Copenhagen

TRACE OF TARGETS:  Brussels Amsterdam Bern Genoa Hamburg Madrid Munich Rome
Berlin Trieste Copenhagen
```

```
# TARGETS:  11

#   #   #   #   #   #   #   #   #   #   #   #   #   #   #   #
Amsterdam (0) TO Bern (3)
DISTANCE:  558
PATH:  Amsterdam > Bern

TRACE OF TARGETS:  Brussels Hamburg Paris Munich Bern
# TARGETS:  5

#   #   #   #   #   #   #   #   #   #   #   #   #   #   #   #
Warsaw (18) TO Warsaw (18)
DISTANCE:  0
PATH:  Warsaw

TRACE OF TARGETS: Warsaw
# TARGETS:  0
#   #   #   #   #   #   #   #   #   #   #   #   #   #   #   #
London (20) TO Dublin (19)
DISTANCE:  ?
PATH:  SORRY - can't reach destination city from start city

TRACE OF TARGETS: ...                 [show me what these are – let it happen as normal]
# TARGETS:  ...                       [show me what this is – let it happen as normal]
```

<u>NOTES regarding TheLog file</u>

- USE THE EXACT FORMAT SHOWN ABOVE, including spacing
- Use Courier New Font & size 7 & smaller margins to print this out in WordPad to get a nice easy-to-read Matrix WITHOUT WRAPAROUND
- The . . . parts would be filled in, of course
- PATH must show cities from START-to-DESTINATION (even though it's easier to show DESTINATION-to-START)
- regarding TRACE OF TARGETS
  - startCity is not in the trace because it's selected during initialization before the while loop
  - the wrap-around (seen above) happens during printing of TheLog file in WordPad
    - the program code just writes a single long line of cities
  - the cities MUST appear in THE ORDER IN WHICH THEY WERE SELECTED. Do NOT just write them out based on the INCLUDED array (which would show all the selected target cities in city-number order instead).

************************************ **NOTES** ************************************

You MUST:

- use the program, class, object, method names described in the specs for easier readability/maintainability among everyone involved in the project
- put the 2 programs and 3 classes in physically separate files
- use OOP for DrivingApp
- [OOP is optional for SetupUtility, but it must be modular. It's just a utility – so it can open/close and read the input file and write out the output files directly without using Map or UI classes].
- use an OOP approach for the 3 classes – so
  - store class-related variables as instance variables in the class (rather than passing a lot of parameters around within the class)
  - the methods inside the 3 classes are instantiable methods, not static methods
  - appropriately specified methods as public or private
  - information hiding, truth in advertising for naming, modularization, …
- have UI house all handling of TransData & TheLog files (open/read/write/close)
- have DrivingApp's Map class access the ?Roads.bin file, not the linear MapData.txt file
- have Map class access the ?Roads.bin FILE on the FILE – it is NOT LOADED INTO MEMORY
- have Map class access N and CityNames in MEMORY and NOT on the FILE for such things as whatsCityName and whatsCityNumber. This data is loaded into memory (in Map's storage) from the files when the constructor runs (though the constructor code would call loadCityNameArray to do the actual work for that part)
- use MY ALGORITHM IMPLEMENTATION for Dijkstra's Shortest Path Algorithm
- use the 3 TYPES OF VALUES in the Adjacency Matrix (as described above and in class)
- use an EXTERNAL Adjacency Matrix (NOT an internal one, and NOT Adjacency Lists)
- do Trace of Targets
  - Even though it's not part of the ALGORITHM handout
  - It print targets AS THE TARGETS GET SELECTED – and NOT just a printout LATER of all the nodes that were INCLUDED

Check your output answers (including path and trace of targets) for REASONABLENESS

- What's the shortest path would be (approximately) looking at the physical map (though use caution for paths going near the Alps and other mountainous or lake-filled areas)
- every city in PATH will be in the TRACE (except START)
- many cities in TRACE may NOT be in the PATH (especially when DESTINATION is a ways from START)
- the order of cities in TRACE will be in increasing distance from START

2 different COSTS are of interest to in a problem like this:

1. the cost of the actual SOLUTION - i.e., the minimum cost path DISTANCE
2. the cost of FINDING the solution - i.e., # TARGETS